UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO FIN DE GRADO

Aplicación Android para la gestión de eventos en una competición deportiva

Gabriel Quintairos Rial
Tutor: Luis Fernando Lago Fernández

Julio 2017

Aplicación Android para la gestión de eventos en una competición deportiva

AUTOR: Gabriel Quintairos Rial
TUTOR: Luis Fernando Lago Fernández

Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2017

Resumen

En la actualidad, son muchas las personas que tienen un smartphone o una tablet y usan diariamente este tipo de dispositivos. Es por ello por lo que existe un gran número de aplicaciones móviles que tienen diversas utilidades, desde los conocidos servicios de mensajería o redes sociales hasta otras aplicaciones que son menos conocidas.

Muchas de estas aplicaciones y muchos de estos dispositivos están siendo aplicados al deporte. Hay aplicaciones que utilizando el GPS te permiten tener controlados tus entrenamientos o incluso registrar tu pulso y tu ritmo cardíaco.

En este Trabajo de Fin de Grado se desarrolla una aplicación móvil para poder gestionar los eventos que ocurren en una competición deportiva. En concreto, esta aplicación permitirá llevar un registro de todos aquellos acontecimientos que ocurran durante un partido de fútbol: goles, tarjetas, sustituciones y otras incidencias.

La primera idea de esta aplicación es que pueda ser utilizada por el equipo arbitral encargado de dirigir un partido, sustituyendo y simplificando las anotaciones manuales que se realizan actualmente. A su vez, también permitirá la generación de un informe del partido cuando este haya terminado.

Además de esto, la aplicación también podrá ser utilizada por cualquier otro agente que necesite llevar un recuento de los eventos que sucedan en un partido, como por ejemplo un periodista que desee realizar una crónica informativa del mismo.

Para realizar este proyecto, se ha utilizado el framework Ionic, explorándose sus utilidades y capacidades a la hora de crear una aplicación móvil multiplataforma. Es por ello por lo que este proyecto ha comenzado con una familiarización y aprendizaje previos de dicha tecnología.

La aplicación desarrollada ha sido satisfactoria y ha cumplido con éxito los objetivos que se habían planteado inicialmente. Para ello se han realizado pruebas en partidos reales y se ha comprobado de este modo su funcionalidad.

Palabras clave

Ionic, aplicación, evento, móvil, dispositivo

Abstract

Nowadays, smartphones and tablets are widely used in everyday life. That is why there exist many mobile applications that provide utilities, from messaging services or social networks to other applications that are less known.

Many of these applications and many of these devices are being applied to sports. There are applications that, using GPS technology, allow you to control your workouts or even record your pulse and your heart rate.

In this Bachelor Thesis, a mobile application to manage the events that occur during a sports competition is developed. Specifically, the application will allow to keep a record of all the events that occur during a football match: score, cards, substitutions and other incidents.

The main goal is that the application can be used by the referee team responsible for running the match, replacing and simplifying the manual annotations that are currently performed. It will also allow the generation of a report when the match is over.

Additionally, the application could also be used by any other agent who needs to keep a record of the events that occur during a football match, such as a journalist who wants to make an informative report.

To accomplish the project, we have used the Ionic framework, exploring its utilities and capacities when creating a multiplatform mobile application. For this reason, the project has started with a familiarization and prior learning of this technology.

The final application is satisfactory and has successfully met the initial objectives. Extensive validation has been carried out during real matches, showing the functionality of the application.

Keywords

Ionic, application, event, mobile, device

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

El objetivo de este Trabajo de Fin de Grado es realizar una aplicación móvil que permita gestionar los eventos que ocurren en una competición deportiva, en concreto, en un partido de fútbol. El proyecto fue inicialmente planteado como una aplicación Android y es por ello por lo que el título así lo indica. Finalmente y tras una investigación de las tecnologías y herramientas disponibles para hacerlo, se optó por realizar una aplicación multiplataforma. Para el desarrollo del proyecto se utilizará el framework Ionic [1], con lo cual también servirá como una introducción y aprendizaje de dicha herramienta.

A continuación, se explican la motivación y los objetivos de este proyecto y se ofrece una descripción de la organización de este documento.

1.1 Motivación

Hoy en día existen numerosas aplicaciones móviles con múltiples funcionalidades. Muchas de ellas nos ayudan en nuestra vida diaria, simplificando nuestras actividades cotidianas. También se ha producido la sustitución de muchos elementos tradicionales por aplicaciones móviles, como es el uso de GoogleMaps [2] en lugar de mapas físicos y los billetes electrónicos o e-tickets en aplicaciones como PassWallet [3] o PassBook [4] en lugar de las entradas físicas a espectáculos. Otro de los próximos elementos a sustituir es el dinero físico, ya que hay aplicaciones como ApplePay [5] o Bizum, la aplicación lanzada en nuestro país que permite realizar pagos a través del teléfono móvil [6], cuyo uso se está comenzando a generalizar.

Además de estas aplicaciones también existen otras orientadas a los deportes. Algunas de ellas te permiten utilizar los sensores del teléfono para medir tus pulsaciones o la distancia que recorres en tu entrenamiento. Otras permiten llevar en el dispositivo un registro del marcador de un partido o de los golpes dados en una pista de golf. Nuestra aplicación será de este último tipo.

Actualmente, para registrar los eventos que suceden en un partido de fútbol (goles, sustituciones, tarjetas, etc.) se realizan anotaciones manuales en una plantilla similar a la que podemos ver en la Figura 1-1. Realizar este proceso puede ser engorroso e incómodo y puede dar lugar a errores al escribir rápido o con prisa. Además, la plantilla en papel puede quedar inutilizada si se moja por la humedad del sudor o de la lluvia.

Estos posibles errores pueden influir posteriormente a la hora de generar un informe o crónica del partido ya que la información que tendremos puede no ser la correcta. En competiciones menores donde no hay medios informativos ni más personas llevando el registro de lo que sucede en un partido, dichos errores pueden ser imposibles de resolver.

Es por ello por lo que se plantea desarrollar una aplicación móvil que permita sustituir las tradicionales anotaciones físicas que se realizan a mano sobre las incidencias que ocurren en un partido de fútbol.

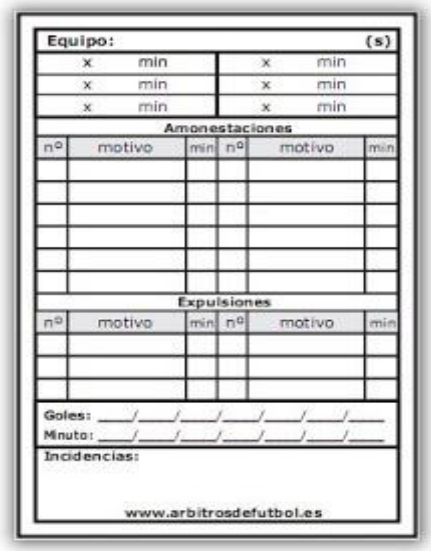

Figura 1-1: modelo de plantilla utilizada en las anotaciones manuales

1.2 Objetivos

El objetivo principal de este Trabajo de Fin de Grado será realizar una aplicación móvil que permita gestionar las incidencias que ocurren en un partido de fútbol. Dicha aplicación deberá permitir además generar un informe de lo que ha ocurrido en el partido cuando éste haya finalizado. Se espera que esta aplicación pueda sustituir a las anotaciones manuales que se realizan actualmente.

Como paso previo a la realización del proyecto y como se verá en uno de los próximos capítulos de esta memoria, se realizó una investigación sobre las tecnologías existentes para realizar desarrollo de aplicaciones móviles. Este proyecto fue concebido con el objetivo de realizar una aplicación Android, aplicando lo aprendido en la asignatura Desarrollo de Aplicaciones para Dispositivos Móviles. Como resultado de la investigación realizada, se descubrió una herramienta interesante y diferente que no había sido explorada anteriormente. Esta herramienta es el framework Ionic y como se comentará más adelante permite crear aplicaciones móviles bajo programación web. De este modo, con un solo proceso se puede crear una aplicación compatible con sistemas operativos diferentes.

Se considerará que los objetivos se han cumplido y que el proyecto ha terminado exitosamente si se consigue una aplicación móvil que sea funcional y que sustituya a los medios tradicionales empleados actualmente.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Estado del arte:** en este capítulo se analizarán otras posibles aplicaciones similares o parecidas que nos puedan servir como punto de partida o como orientación a la hora de comenzar nuestro proyecto.
- **Definición del proyecto:** aquí se estudiarán el alcance de la aplicación, la metodología, herramientas y tecnologías utilizadas en su desarrollo. Además se realizará una recopilación de los requisitos funcionales y no funcionales que debe cumplir nuestro proyecto.
- **Diseño:** en este apartado se hablará de las clases internas del proyecto, del diseño de la interfaz gráfica de la aplicación y de algunas características esenciales que deba cumplir.

- **Desarrollo:** en este apartado abordaremos la forma en la que se ha llevado a cabo el proyecto una vez finalizada la fase de diseño. Tratará de todo el proceso de codificación y las eventualidades que haya habido en él.
- **Integración, pruebas y resultados:** durante el desarrollo del proyecto se realizaron pruebas unitarias de los módulos que se iban creando, así como pruebas finales una vez estuvo listo el primer prototipo. Cuando se consideró la aplicación finalizada, se realizaron pruebas de validación y se utilizó en el ámbito real donde se espera que sea utilizada. Todas estas pruebas y sus resultados se comentarán en este apartado de la memoria.
- **Conclusiones y trabajo futuro:** para finalizar el cuerpo de este documento, se comentarán las conclusiones que se pueden elaborar tanto de la aplicación en sí como del trabajo realizado. A su vez, se comentarán posibles mejoras que se puedan hacer y el trabajo futuro que se puede llevar a cabo.
- **Referencias:** aquí se incluye la bibliografía y referencias que se han tenido en cuenta a la hora de documentar el proyecto y elaborar esta memoria.
- **Glosario:** se incluyen las palabras más importantes en este proyecto y una breve definición de las mismas.
- **Anexos:** entre ellos se encuentran un manual de instalación de la aplicación en un dispositivo Android, un manual para el programador con una explicación para la utilización del Framework Ionic, las maquetas realizadas en la fase de diseño de la interfaz gráfica de la aplicación y algunas capturas de pantallas correspondientes a las ventanas principales de la aplicación.

2 Estado del arte

2.1 Introducción

Desde el lanzamiento de la Play Store (en sus inicios bajo el nombre de Android Market) de Google el 28 de agosto de 2008 [7] y de la App Store de iOS el 10 de julio de 2008 [8] son muy numerosas las aplicaciones para dispositivos móviles Android e iOS que nos podemos encontrar en el mercado. El crecimiento de estas dos plataformas tanto en número de aplicaciones como en número de desarrolladores ha sido altísimo desde su lanzamiento hasta la actualidad.

Las 2.300 aplicaciones que tenía la Play Store en marzo de 2009 [9] se han quedado muy cortas en comparación con los 2,8 millones de ellas que había en marzo de 2017 [10]. A su vez, las 500 aplicaciones con las que empezó la App Store de iOS se han visto ampliamente superadas con los 2,2 millones de ellas que había en enero de 2017 [11].

Aparte de estas dos "tiendas" de aplicaciones en las que se encuentran la mayoría de las que están en el mercado, también existen otras de menor importancia pero que también poseen un gran número de aplicaciones. Son por ejemplo la Windows Store, la Amazon Appstore y la Blackberry World; que en marzo de 2017 contaban con 669.000, 600.000 y 234.500 aplicaciones disponibles para descarga respectivamente [12].

Habiendo tal cantidad de aplicaciones disponibles, nos podemos encontrar casi con todo lo que nos imaginemos. Incluso muchas veces nos encontramos varias alternativas que responden a una misma necesidad y deberemos pararnos a valorarlas para saber cuál se adapta mejor a lo que buscamos.

2.2 Aplicaciones relacionadas con las actividades deportivas

El reducido tamaño de los Smartphone y la existencia de multitud de brazaletes y fundas para portarlos con nosotros hace que estos dispositivos se hayan convertido en nuestros compañeros a la hora de hacer deporte. Además también hay en el mercado novedosos auriculares que funcionan por Bluetooth y que están específicamente adaptados para ser utilizados mientras se realiza una actividad deportiva.

Aparte de utilizar nuestros Smartphones para escuchar música y amenizar nuestro ejercicio físico, también le podemos dar un uso algo menos ocioso y más técnico y recopilar datos relacionados con nuestra actividad. Dentro de esto, podemos encontrarnos con dos tipos de aplicaciones: las que recopilan datos sobre nuestro rendimiento y las que recopilan eventos de la actividad o competición que se está llevando a cabo.

2.2.1 Aplicaciones medidoras de rendimiento y actividad física

Este tipo de aplicaciones son aquellas que nos permiten monitorizar nuestro rendimiento a la hora de entrenar o realizar ejercicio. Dentro de sus funciones pueden estar medir y controlar nuestra frecuencia cardíaca, calcular la distancia y el tiempo que somos capaces de correr o de andar en bici, calcular nuestra velocidad media en un entrenamiento, etc. Como ejemplos de estas aplicaciones podemos encontrar:

- **Runtastic GPS Running Fitness:** esta aplicación utiliza el GPS del dispositivo para registrar nuestras actividades deportivas. Literalmente según su descripción en la Play Store: *"Mide distancia, tiempo, ritmo, calorías quemadas, velocidad, altitud y mucho más."* [13]
- **Nike+ Run Club:** aplicación de la marca comercial Nike que permite registrar y guardar los resultados de carreras, compararlos con los de otros corredores y compartir impresiones y comentarios después de haber participado en una carrera o

competición. Además, esta aplicación ofrece planes de entrenamiento adaptados al rendimiento del usuario. [14]

- **Google Fit:** esta aplicación propiedad de Google nos permite realizar el seguimiento de nuestra actividad física y acceder a los datos y estadísticas generadas en tiempo real. Además recopila datos de otras aplicaciones y también permite controlar nuestra nutrición, horas y calidad del sueño, peso corporal, pasos realizados durante el día y calorías quemadas en un entrenamiento. [15]

2.2.2 Aplicaciones que recopilan información sobre los eventos que ocurren en una actividad deportiva

Por otra parte tenemos este otro tipo de aplicaciones que en vez de registrar nuestro rendimiento físico personal registran aquellas eventualidades que ocurren en nuestra actividad deportiva. Por ejemplo, este tipo de aplicaciones pueden servir para contar el número de golpes que un jugador realiza mientras practica golf o para llevar el marcador en un partido de tenis. Como ejemplos tenemos:

- **Golf scorecard Pro:** como se comentó anteriormente, esta aplicación permite contar el número de golpes realizados en una partida de golf. Permite guardar los datos de entre 1 y 5 jugadores y contabilizar de 9 a 18 hoyos. [16]
- **Estadísticas de baloncesto OmniStats:** como su nombre indica, esta aplicación permite llevar el registro de lo que ocurren en un partido de baloncesto: puntos obtenidos por cada jugador, faltas personales y técnicas realizadas, tiros libres anotados y fallados, puntuación al final de cada cuarto, etc. [17]
- **Tracking Tennis Matches:** aplicación que sirve para llevar el tanteo de un partido de tenis. Permite llevar un recuento de los saques, ruptura de servicio, dobles faltas, puntuación de cada set, etc. Además permite analizar posteriormente cada partido y llevar un control de las estadísticas. Se puede utilizar tanto en partidos individuales como en partidos de dobles. [18]
- **Marcador de fútbol electrónico:** aplicación que sirve para mostrar el marcador de un partido de fútbol en el que puedes ir anotando los goles de cada equipo. También permite imitar el sonido de un silbato. [19]

En este Trabajo de Fin de Grado vamos a realizar una aplicación de este tipo que nos permita recopilar todas aquellas cosas que sucedan en un partido de fútbol. Es por ello por lo que vamos a analizar específicamente las aplicaciones relacionadas con la temática a tratar. Como primera aproximación, si buscamos en la Play Store las palabras "Árbitro" o "Referee" podemos encontrarnos con que nos aparecen multitud de resultados como podemos ver en la Figura 2-1. La mayoría de estos resultados son aplicaciones muy sencillas, que simplemente ponen la pantalla del dispositivo de color amarillo o rojo simulando ser una tarjeta o que imitan el sonido de un silbato. Estas aplicaciones recreativas no tienen ninguna relación con el trabajo que se va a desarrollar.

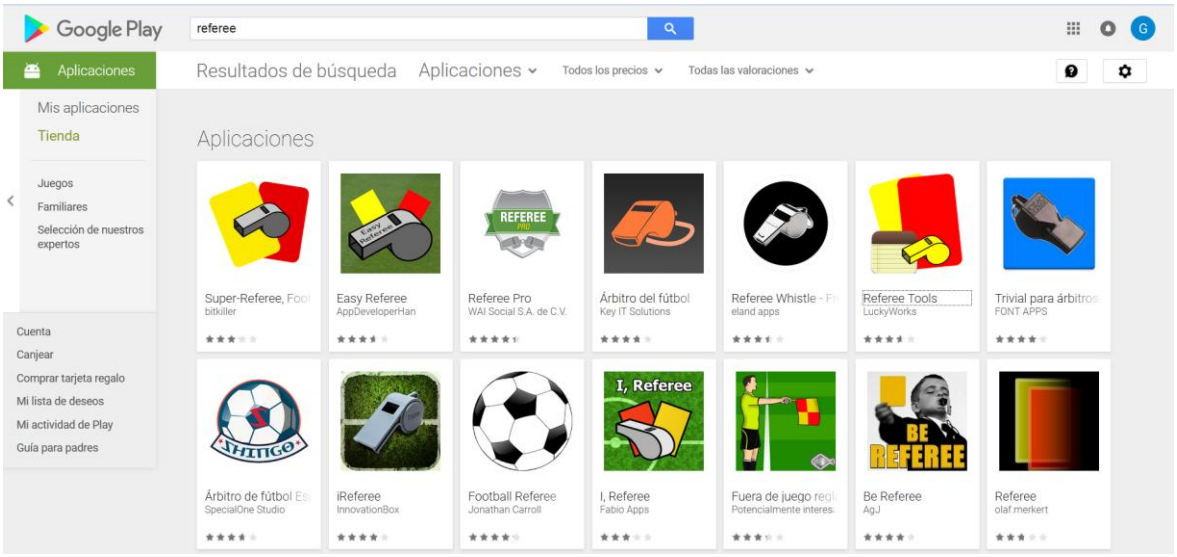

Figura 2-1: Resultados de la búsqueda de la palabra "referee" en la Play Store

Dentro de todos los resultados obtenidos en la búsqueda vamos a hacer un análisis comparativo de las aplicaciones existentes que se parezcan a lo que queremos hacer. Tendremos en cuenta sus funcionalidades y veremos qué aspectos podemos imitar, cuáles podemos mejorar y qué nuevas cosas podemos añadir. A continuación mostramos los cuatro ejemplos más relevantes que hemos encontrado:

- **Referee Pro:** esta aplicación permite crear ligas, realizar una gestión de las designaciones arbitrales y de las actas de partidos de fútbol. Su mayor inconveniente es que no está pensada para ser utilizada durante el desarrollo del juego, sino que más bien está pensada para ser utilizada antes y después del partido. No dispone de un sistema de gestión de eventos en tiempo real, sino que supone que has cubierto ya el acta del partido y luego es donde la aplicación te ayuda a gestionarla. [20]
- **Árbitro de bolsillo:** esta aplicación muestra tarjetas amarillas y rojas poniendo la pantalla del dispositivo de dicho color. También imita un silbato de árbitro. No tiene ningún tipo de gestión o entrada de datos.[21]
- **Cuarto árbitro:** permite la carga de alineaciones antes del partido, incluye un cronómetro para controlar el tiempo del partido (aunque no funciona correctamente y causa problemas al usar la aplicación), permite añadir goles, tarjetas y sustituciones, emite avisos durante el desarrollo del juego, genera un acta interactiva al finalizar el partido. [22]
- **iReferee:** permite imitar el sonido de un silbato, tiene un marcador al que se le pueden sumar goles, contiene un cronómetro, simulador de moneda para el sorteo de campo inicial y permite poner la pantalla de amarillo o rojo simulando una tarjeta. [23]

Veamos en la siguiente tabla un resumen de las funcionalidades de cada una:

Aplicación	Introducción de alineaciones previa al partido	Introducción de goles, tarjetas y sustituciones	Emisión de avisos durante el juego	Introducción de incidencias	Generación de informe al finalizar el partido
Referee Pro	Sí	No	No	No	A medias
Árbitro de bolsillo	No	No	No	No	No
Cuarto árbitro	Sí	Sí	Sí	No	Sí
iReferee	No	A medias	No	No	No

Tabla 2-1: comparativa de la funcionalidad de aplicaciones similares

La aplicación que se va a desarrollar deberá cumplir todas las funcionalidades indicadas en la tabla anterior. Como hemos visto, ninguna de las aplicaciones que hay actualmente las cumple todas. La aplicación que más se parece a lo que queremos hacer es la llamada *Cuarto árbitro*, pero dicha aplicación tiene el inconveniente de no funcionar correctamente al tener incluido un reloj que ralentiza y en ocasiones colapsa la aplicación. En nuestra aplicación no se plantea por el momento incluir un cronómetro, pues consideramos que dicho elemento es más cómodo seguir llevándolo en la muñeca. Además, el informe que genera dicha aplicación es interactivo y nosotros queremos que se genere un PDF para poder enviar o portar el acta por email o por otros medios. Para finalizar, ninguna de las aplicaciones encontradas permite al usuario añadir incidencias.

2.3 Conclusiones

Como hemos podido observar hay muchísimas aplicaciones relacionadas con las actividades deportivas. Dentro de las que están relacionadas con el fútbol, en concreto con el árbitro, encontramos algunas de ellas que son similares a lo que queremos hacer y otras que no tienen nada que ver.

Tras haber analizado el estado de las aplicaciones móviles relacionadas con la temática a tratar, observamos que ninguna de las aplicaciones que hemos encontrado permite realizar todo aquello que queremos. En concreto, ninguna de ellas permite al usuario añadir incidencias. Además tampoco hemos encontrado ninguna que genere un informe en PDF al finalizar el partido usando los datos que el usuario había ido añadiendo durante el mismo.

En este Trabajo de Fin de Grado se realizará una aplicación que tenga un diseño diferente a las ya vistas, que sea más dinámica e intuitiva y que permita generar un informe en PDF a la finalización del partido. Además se harán las suficientes pruebas para evitar que la aplicación tenga errores al usar alguno de sus elementos o que se colapse durante su uso. Dicha aplicación también debe permitir al usuario añadir incidencias que ocurren en un partido y que sean relevantes, como por ejemplo lesiones graves de jugadores o incidentes ocurridos con el público del partido.

3 Definición del proyecto

3.1 Introducción

En este apartado de la memoria se definirá el proyecto y lo que vamos a hacer en este Trabajo de Fin de Grado. En concreto se estudiará su alcance, la metodología, tecnologías y herramientas empleadas durante su desarrollo y el detalle de sus requisitos funcionales y no funcionales.

3.2 Alcance

Este Trabajo de Fin de Grado tiene el propósito de realizar una aplicación que sea utilizada por árbitros, informadores y cualquier otra persona que necesite realizar un informe de un partido de fútbol.

Dentro de su alcance estará la gestión de los datos recogidos por el usuario a lo largo de la competición, la generación de avisos de interés relacionados con el encuentro y la posterior generación del informe.

3.3 Metodología

A la hora de decidir cómo se iba a desarrollar esta aplicación se ha decidido seguir un desarrollo en cascada [24]. A pesar de esto, dicha metodología no se ha seguido estrictamente y se ha optado por mezclarla con un desarrollo ágil [25] que permitiese incorporar nuevas funcionalidades mientras se iba codificando. Esto responde a la necesidad de volver a codificar después de la fase de pruebas.

El desarrollo en cascada se basa en establecer un orden riguroso entre las etapas que va a seguir el proyecto. Al final de cada una de las etapas se realiza una revisión para ver si se han cumplido los objetivos propuestos para ella. Además, cada una de las etapas no puede comenzar si la anterior no ha finalizado.

El desarrollo ágil es iterativo e incremental. En él se realizan iteraciones del ciclo de vida en las cuales se realizan todas las fases del proyecto (análisis, diseño, codificación y pruebas). Al finalizar cada iteración el producto está listo para ser usado. Dicho producto se conocerá como un *prototipo*. En las iteraciones siguientes se volverá a repetir el proceso y se podrán modificar los requisitos previos. Los siguientes prototipos tendrán cada vez mayor funcionalidad y serán más completos hasta que en la última iteración se llegue al producto final.

En nuestro proyecto se ha decidido realizar algo intermedio, siguiendo un desarrollo en cascada como patrón principal pero permitiendo una metodología más ágil entre las fases de codificación y pruebas. Las fases de definición del proyecto, análisis y diseño se han realizado siguiendo una metodología en cascada de forma estricta. Una vez llegada la fase de codificación y pruebas, se ha desarrollado un prototipo inicial que fue probado. Como consecuencia de cada periodo de pruebas, se volvía a la fase de codificación y se modificaban aspectos que no eran usables o tenían algún error o se añadía alguna funcionalidad nueva que se nos ocurría que podría ser interesante mientras probábamos el prototipo. Este ciclo entre las fases de codificación y pruebas se realizó varias veces creando en cada una de ellas un prototipo más completo que el anterior hasta llegar a la última iteración en la que se generó el producto final. Todo esto será comentado más en detalle en los siguientes subapartados.

Las diferentes fases de esta metodología con sus correspondientes tareas han sido las siguientes:

1. **Definición del proyecto y su alcance:** en primer lugar se ha acotado la idea principal que motivó la realización de este Trabajo de Fin de Grado, estableciendo su alcance y los objetivos que se pretenden alcanzar al finalizar el proyecto.
2. **Análisis de requisitos:** tras haber definido el proyecto, nos centramos en los requisitos funcionales y no funcionales que debe cumplir la aplicación para que sea exitosa.
3. **Investigación de las posibles herramientas y tecnologías a emplear:** una vez supimos los requisitos que debía cumplir la aplicación realizamos una investigación para ver con qué herramientas podíamos desarrollarla.
4. **Diseño de la aplicación (clases a emplear, interfaz):** comenzamos haciendo un diagrama de clases para la lógica interna de la aplicación y bocetos para el diseño de la interfaz gráfica.
5. **Codificación:** se comienza la programación de la aplicación teniendo en cuenta los requisitos que debe cumplir y el diseño realizado en las anteriores fases.
6. **Pruebas unitarias:** se van probando las clases de la aplicación en solitario viendo si cumplen los requisitos que se les piden. Si alguna de las pruebas nos muestra errores o descubrimos una posible mejora o nueva funcionalidad, volvemos a la fase anterior. Las pruebas serán comentadas más en detalle en otro de los apartados de esta memoria.
7. **Pruebas de validación y verificación:** tras haber finalizado la aplicación, se harán pruebas de la misma en partidos reales y se corregirán los errores que se detecten o se mejorará la usabilidad de la misma. Por tanto se establece un ciclo entre las fases 5, 6 y 7 que está relacionado con el desarrollo ágil que estamos siguiendo en este proyecto. Estas pruebas también serán comentadas en otro de los apartados de la memoria.
8. **Pruebas de compatibilidad:** al ser una aplicación multiplataforma se han realizado pruebas con diferentes sistemas operativos para estudiar la compatibilidad de la aplicación con ellos. También se ha probado la compatibilidad con diferentes tamaños de pantalla.
9. **Puesta en marcha de la aplicación:** una vez se hayan completado exitosamente todas las pruebas previstas, la aplicación se pondrá en marcha y se utilizará en sustitución de las anotaciones manuales.

El ciclo de vida terminaría con una última fase de mantenimiento de la aplicación desarrollada. Dicho mantenimiento se seguirá realizando, pero ya no será objeto de este Trabajo de Fin de Grado, sino que se realizará posteriormente una vez que la aplicación esté en funcionamiento.

3.4 Tecnologías y herramientas

Este proyecto fue pensado primeramente para ser realizado en Android Studio [26] al igual que la asignatura de Desarrollo de Aplicaciones para Dispositivos Móviles. Tras una primera fase de investigación, fue descubierto el Framework Ionic. Esta herramienta permite realizar aplicaciones móviles para Android [27], iOS [28], Blackberry [29] y Windows Phone [30] programando tan sólo una versión de la misma. Se caracteriza por utilizar código web en la codificación y tener librerías que lo adaptan al sistema operativo del Smartphone en el que se vaya a utilizar. Por este motivo, se descartó la utilización de Android Studio y se optó por utilizar el Framework Ionic. Esta decisión fue tomada en base a la gran ventaja que supone el hecho de que la aplicación sea multiplataforma así como la

posibilidad que se ofrece de conocer y aprender una nueva herramienta. A continuación se detallan las tecnologías y herramientas utilizadas:

3.4.1 Tecnologías

Ionic es una herramienta que utiliza las siguientes tecnologías:

- JavaScript [31]: es un lenguaje de programación web opensource que interactúa con HTML y permite realizar páginas web interactivas. Es interpretado por el navegador y no necesita compilador y a diferencia de Java no es necesariamente orientado a objetos.
- TypeScript [32]: es un lenguaje de programación web de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript que esencialmente añade tipado estático y objetos basados en clases.
- HTML [33]: es un lenguaje que se escribe utilizando etiquetas y que sirve para la realización de páginas web.
- CSS [34]: es un lenguaje de diseño gráfico muy usado para establecer el diseño visual de las páginas web.

3.4.2 Herramientas

Las siguientes herramientas fueron utilizadas en este proyecto:

3.4.2.1.1 Desarrollo y codificación

- Ionic: es una herramienta gratuita y opensource que permite realizar aplicaciones móviles multiplataforma utilizando HTML, CSS y JavaScript.
- Sublime Text [35]: editor de texto que permite remarcar la sintaxis de múltiples lenguajes, entre ellos los utilizados en este Trabajo de Fin de Grado, haciendo que sea más fácil editar el código fuente.

3.4.2.1.2 Diseño de las maquetas

- Mocking Bot [36]: es una herramienta online y gratuita que te permite realizar maquetas y prototipos de aplicaciones móviles de una forma sencilla

3.4.2.1.3 Control de versiones

- GIT [37]: es un software diseñado para el soporte de control de versiones. Es ampliamente utilizado en numerosos proyectos.
- Bitbucket [38]: plataforma gratuita que trabaja bajo el software GIT y que permite crear repositorios privados y tenerlos en la nube, gestionando los cambios que se producen en ellos.

3.4.2.1.4 Pruebas

Para realizar las pruebas se utilizaron los navegadores Internet Explorer [39], Mozilla Firefox [40] y Google Chrome [41] para las pruebas hechas en el ordenador y dos dispositivos móviles para las pruebas hechas en dispositivo físico una vez fue generada la apk [42]. Dichos dispositivos móviles son un Vodafone Smart Ultra 6 y un Sony Xperia 5.

Al no disponer físicamente de dispositivos iOS, Blackberry ni Windows Phone, las pruebas correspondientes a estos dispositivos se hicieron en emuladores instalados en el ordenador. Para las pruebas en tablet se ha usado un dispositivo físico Neo 3 Lite.

3.4.2.1.5 Bibliotecas y librerías

- jspdf [43]: librería de JavaScript utilizada para crear documentos PDF. Será utilizada en la aplicación para generar el informe al finalizar el partido.
- AngularJS [44]: es un framework de JavaScript de código abierto y mantenido por Google que se utiliza para realizar aplicaciones web con capacidad Modelo-Vista-Controlador (en adelante MVC).
- NodeJS [45]: entorno de código abierto implementado en JavaScript que funciona en la parte del servidor y en concreto es muy útil a la hora de implementar servidores web.
- Cordova [46]: entorno de desarrollo de aplicaciones móviles que permite encapsular código HTML, CSS y JavaScript y adaptarlo a diversos sistemas operativos móviles. Las aplicaciones realizadas bajo este entorno son híbridas, pues son una mezcla entre aplicación móvil y aplicación web, sin llegar a ser de uno u otro tipo exclusivamente.

3.5 Requisitos

En este subapartado se definirán los requisitos que debe cumplir la aplicación y las funcionalidades que se pretenden abarcar con ella. Los requisitos están divididos en dos tipos: los requisitos funcionales y los requisitos no funcionales. Los primeros tratan sobre el comportamiento y las funcionalidades esperadas de la aplicación y los segundos miden su calidad y usabilidad.

3.5.1 Requisitos Funcionales

RF 1: Los usuarios podrán crear nuevos partidos introduciendo el nombre de dos equipos y la categoría de dicho partido.

RF 2: En cada partido tendrá que estar seleccionada una hora de inicio de la primera parte y una hora de inicio de la segunda parte.

RF 3: El partido estará formado por dos equipos, cada uno de los cuales deberá tener un mínimo de 7 jugadores titulares y un máximo de 11 y un máximo de 5 suplentes.

RF 4: El usuario podrá añadir jugadores a cada alineación. Cada jugador deberá tener un nombre y un dorsal, que será un número natural entre el 1 y el 99. Dos jugadores del mismo equipo no podrán llevar el mismo dorsal. Cada jugador podrá ser titular, capitán o portero.

RF 5: Dentro de cada alineación deberá haber un portero y un capitán entre los jugadores titulares.

RF 6: A cada jugador titular se le podrá asignar un gol. Cada gol deberá ser asignado en un minuto válido del partido, es decir, entre el 1 y el 90. El gol podrá ser normal, de penalti o en propia meta.

RF 7: Cada jugador titular podrá ser sustituido por un jugador suplente. Cada sustitución deberá tener un minuto válido (entre el 1 y el 90) y el jugador titular dejará de serlo y el suplente pasará a ser titular.

RF 8: A cada equipo se le podrán asignar oficiales técnicos. Cada oficial deberá tener un nombre y un puesto (delegado, delegado de campo, entrenador, segundo entrenador, médico, fisioterapeuta, utillero, entrenador de porteros). Dentro de un mismo equipo no podrá haber dos oficiales con el mismo puesto.

RF 9: A cada jugador u oficial técnico se le podrán asignar tarjetas. Cada tarjeta deberá tener un minuto válido (entre el 1 y el 90), un motivo y un color (amarilla o roja).

RF 10: La aplicación deberá notificar mediante un aviso por pantalla cuando a un jugador u oficial técnico se le asigna una segunda tarjeta amarilla.

RF 11: Si a un jugador se le asigna una tarjeta roja, será expulsado. A un jugador que ha sido expulsado no se le podrán asignar goles ni podrá formar parte de una sustitución en un minuto posterior al de su expulsión.

RF 12: La aplicación permitirá registrar incidencias que sucedan en el partido. Cada incidencia tendrá un tipo (lesión, desperfecto en las instalaciones, fallo en la iluminación, etc.) y una descripción en campo de texto.

RF 13: Al finalizar el partido, la aplicación permitirá generar un informe en el que se muestren los eventos que han sucedido.

3.5.2 Requisitos No Funcionales

RNF 1: La aplicación será multiplataforma. Será compatible con las siguientes versiones o superiores: Android 4.1, iOS 8, Blackberry OS 10, Windows Phone 8.1.

RNF 2: La aplicación está pensada para uso en smartphones o tablets que cumplan el requisito anterior en cuanto a su sistema operativo.

RNF 3: La aplicación hará uso de la pantalla táctil del dispositivo como método de interacción con el usuario.

RNF 4: Los avisos que tenga que mostrar la aplicación al usuario se mostrarán en pantalla por medio de *alerts*.

RNF 5: El sistema no requerirá paradas programadas para su mantenimiento.

RNF 6: La aplicación no necesitará conexión a internet para su funcionamiento.

4 Diseño

4.1 Introducción

En este capítulo se tratarán las decisiones de diseño tomadas para desarrollar la aplicación definida en el capítulo anterior. En concreto estudiaremos el patrón escogido para la arquitectura de la aplicación, el diseño de sus clases y el diseño de su interfaz gráfica.

4.2 Arquitectura de la aplicación

Originalmente se pensó en el tradicional modelo de arquitectura MVC [47] para nuestra aplicación. Este modelo es muy fácil de implementar en aplicaciones Android, pues se proporcionan los métodos y las características adecuadas (fragments, layouts, clase View, activities, etc.) para llevarlo a cabo. Sin embargo, al utilizar Ionic y aplicaciones basadas en AngularJS, este modelo sufre una ligera variación. El modelo con las variaciones correspondientes es llamado Modelo Vista VistaModelo (en adelante MVVM) [48]. La diferencia radica en que la interacción entre la vista y el controlador será en los dos sentidos. El controlador muestra los datos en la vista y si en la vista hay un cambio de datos, se actualiza el modelo automáticamente. Veamos cómo funciona esto en una aplicación basada en AngularJS:

- **Modelo:** es la parte de la aplicación encargada de tratar y de manejar los datos básicos con los que se trabaja. En nuestro caso, dichos datos corresponderán a todo aquello que el usuario introduzca por la pantalla en cada partido: alineaciones, nombres de los técnicos, goles, tarjetas, sustituciones, incidencias, etc. Para trabajar con ello, nuestra aplicación utilizará un *provider* [49], esto es, un servicio que permita manejar datos, almacenarlos y acceder a ellos. Dicho servicio estará implementado en JavaScript y los datos serán manejados con estructuras propias de este lenguaje como Arrays, HashMaps y objetos.
- **Vista:** su función se corresponde con la visualización de los datos contenidos en el modelo y en nuestra aplicación será la interfaz gráfica. Estará formada por ficheros HTML y CSS. En esta aplicación se ha optado por utilizar un fichero de cada tipo para cada una de las pantallas. Las pantallas activas son puestas en una pila y la aplicación puede ir navegando por ella, permitiendo de esta forma navegar hacia delante y hacia atrás entre las diversas pantallas a la vez que quita o apila un elemento en la pila.

Tras haber visto las dos partes que forman nuestra aplicación, ahora estudiaremos la interacción entre ellas. Esta es la principal diferencia con el modelo MVC, ya que la interacción bidireccional entre las dos partes es la que sustituye al controlador.

Dicha interacción sucede por medio de llamadas a funciones programadas en código JavaScript. Al igual que sucede en una página web, el código es ejecutado en el lado del cliente, en este caso en la propia aplicación. Aquí distinguimos dos casos:

1. **El usuario interactúa con la vista:** esto sucede cada vez que el usuario utiliza la pantalla táctil del dispositivo para introducir o modificar información. Al hacer esto, se realizan llamadas a funciones de JavaScript encargadas de actualizar el modelo. Dichas funciones se encargan de gestionar la entrada del usuario (controlar los posibles errores o entradas inválidas) y trasladarla al *provider* correspondiente donde se encuentren los datos que se van a modificar

o donde esté el array o clase a la que se le añadirán datos nuevos. De esta forma, nuestra estructura de datos alojada en el *provider* queda actualizada.

2. **El modelo se actualiza o cambia de estado:** cada vez que se produzca un cambio de estado o una actualización del modelo que deba ser mostrada en pantalla, se produce una interacción con la vista. Esto sucede cuando la aplicación debe mostrar algún aviso (por ejemplo que la tarjeta amarilla introducida por el usuario es la segunda y que el jugador debe ser expulsado) o cuando se actualiza alguna estructura de datos (se añade un jugador y ahora la alineación tiene un jugador más). Cuando esto pasa, el *provider* realiza una llamada a una función JavaScript que se ejecutará en la vista y mostrará en la pantalla la información correspondiente.

En nuestra aplicación tanto el *provider* como la vista se encuentran almacenados de forma local y todos estos procesos ocurren en el lado del cliente. No obstante, el *provider* podría estar almacenado en un servidor y que la comunicación sucediese utilizando HTTP para realizar la transferencia de información. Esto es bastante habitual en las aplicaciones de este estilo y en concreto en esta arquitectura utilizada. NodeJS se suele utilizar del lado del servidor y AngularJS del lado del cliente y al estar ambos implementados en JavaScript es muy fácil hacer que funcionen juntos.

4.3 Diagrama de clases

A continuación se muestra el diagrama de clases correspondiente al modelo de nuestra aplicación:

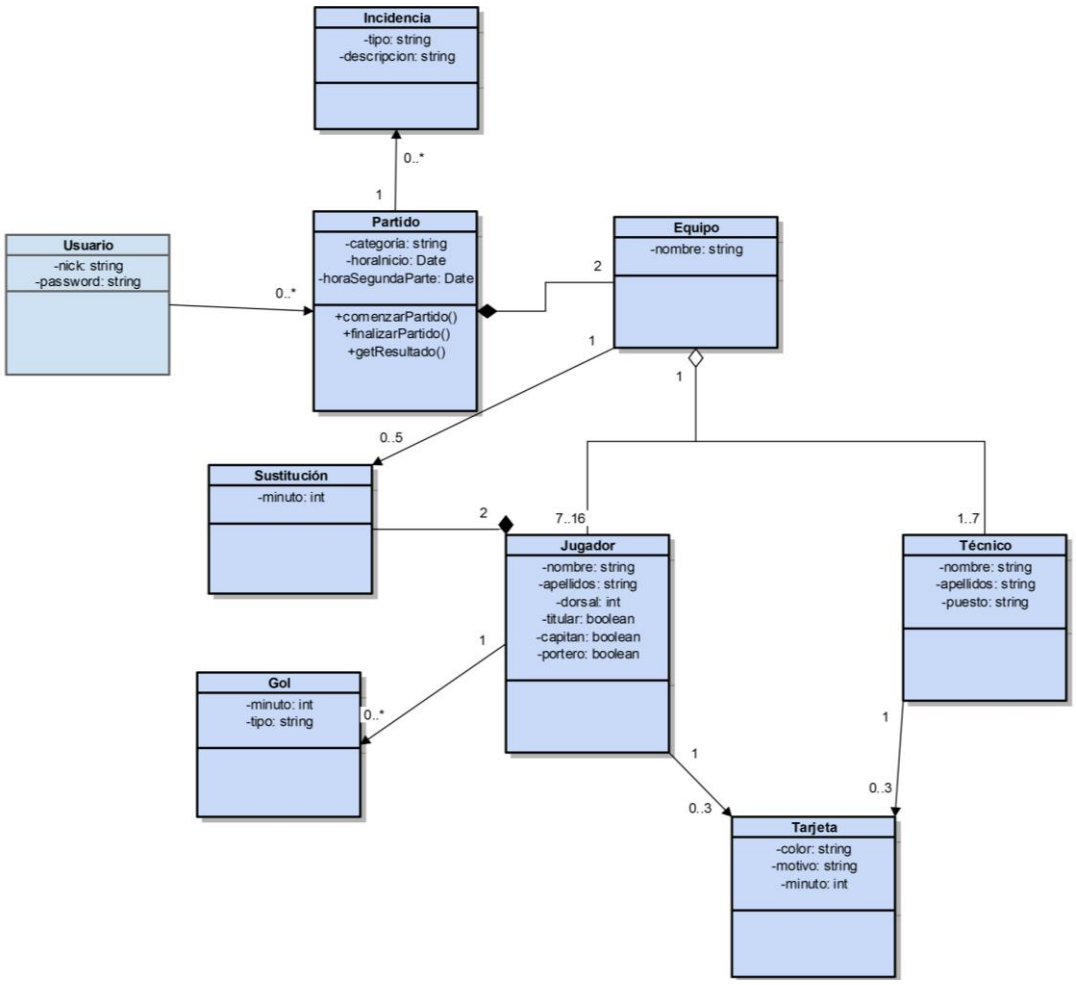

Figura 4-1: diagrama de clases de la aplicación

Podemos ver en este diagrama como se encuentran representadas todas aquellas entidades que serán importantes en nuestra aplicación. Vamos a proceder a explicarlo intentando ir de izquierda a derecha:

- **Usuario:** será el usuario principal de la aplicación. Tendrá un nick y un password por si se quiere implementar posteriormente un sistema de registro e inicio de sesión. La aplicación está pensada para que cada usuario pueda tener un único partido abierto, pero en el diagrama de clases se deja la opción a que en un futuro pueda tener más de uno en un historial.
- **Partido:** será la clase principal de la aplicación y englobará a todas las demás. Como atributos propios tendrá su categoría correspondiente, su hora de inicio y su hora de inicio de la segunda parte. Además, dicha clase será la que contenga a ambos equipos y las incidencias que se produzcan.
- **Equipo:** cada uno de los dos que participan en el partido tendrán una clase en nuestra aplicación. Cada equipo vendrá determinado por su nombre. Esta clase contendrá la alineación de jugadores y el listado de técnicos que se presenten al partido.
- **Jugador:** será cada uno de los participantes en el partido. Cada jugador podrá ser titular o suplente, portero o jugador de campo y podrá ser o no capitán. Además deberá tener nombre y un dorsal. Cada equipo deberá tener entre 7 y 16 jugadores.
- **Técnico:** serán los oficiales que estén en el banquillo de cada equipo. Cada uno tendrá un nombre y su puesto correspondiente.
- **Gol:** cada jugador titular podrá tener asignados goles. Cada gol se marcará en un minuto y será de un tipo: normal, de penalti o en propia meta.
- **Tarjeta:** las tarjetas podrán ser asignadas a los jugadores o técnicos. Cada tarjeta deberá ser registrada con el minuto en el que se ha sacado, su motivo y el color de la misma.
- **Sustitución:** esta clase es algo especial, pues engloba a dos jugadores y a su vez pertenece al equipo que la realiza. Cada sustitución deberá ser registrada con el minuto en el que se ha realizado. Cada equipo tendrá un límite de sustituciones según la categoría del partido.
- **Incidencia:** cada incidencia será de un tipo (lesión, problemas en las instalaciones, suspensión del partido, etc.) y tendrá una descripción. Las incidencias quedarán englobadas dentro del partido.

Los métodos getters y setters se han omitido en este diagrama por simplicidad.

4.4 Interfaz gráfica

A la hora de pensar en la interfaz gráfica de la aplicación, se realizaron maquetas con la herramienta *Mocking Bot* que representasen lo que se pretendía lograr. Dichas maquetas sirvieron como una guía a la hora de comenzar a programar las vistas y fueron una referencia para ver qué se quería hacer. El resultado final no se ajusta fielmente a ellas, pues se modificaron aspectos de diseño, se cambiaron cosas y se añadieron y eliminaron algunos elementos.

Uno de los principales motivos por los que las maquetas no se ajustan al resultado final es que se realizaron pensando en una implementación de la interfaz utilizando Android Studio, es decir, extendiendo la clase View de Android y utilizando los XML y Layouts que ofrece dicho entorno. Al realizar finalmente el desarrollo utilizando Ionic, el aspecto ha cambiado, ya que la herramienta utilizada tiene sus propios elementos (botones, campos de texto, pestañas, etc.) y el diseño gráfico utilizando HTML y CSS es bastante diferente en cuanto a la presentación de los elementos.

Las maquetas realizadas con la herramienta mencionada se pueden consultar en el Anexo C de este documento.

5 Desarrollo

5.1 Introducción

En este apartado se tratarán los aspectos más importantes del desarrollo de nuestra aplicación. Tras las fases de definición del proyecto, análisis y diseño comentadas anteriormente, la siguiente fase ha sido la de codificación. Esta ha sido la fase del proyecto con mayor cantidad de investigación y aprendizaje por el hecho de realizarse con nuevas herramientas nunca antes utilizadas.

Para poder utilizar el framework Ionic es necesario instalar varias librerías y herramientas auxiliares en nuestro ordenador. Todas estas cuestiones vendrán comentadas en el Manual del desarrollador en el Anexo B de este documento. A continuación se comentará la estructura del proyecto

5.2 Estructura del proyecto

La estructura de la aplicación realizada se corresponde con la estructura habitual de una aplicación basada en Cordova por ser el entorno en el que funciona Ionic. Vamos a comentarla con detalle pues ha sido uno de los elementos más novedosos de la fase de codificación. A continuación se muestra una imagen correspondiente al directorio donde se encuentra el código de nuestra aplicación:

Figura 5-1: estructura del proyecto

- **hooks:** en esta carpeta se sitúan aquellos scripts que sea necesario ejecutar durante la compilación de la aplicación. Al no tener ningún script de este tipo en nuestro proyecto, dicha carpeta únicamente contiene un fichero README.md generado automáticamente al crear el proyecto.
- **node_modules:** aquí se encuentran todas las librerías que instalamos en nuestra aplicación usando NodeJS y su instrucción *npm install* además de las que ya vienen por defecto al crear el proyecto. La única librería adicional que hemos añadido y que se encuentra en esta carpeta es la librería *jspdf*.
- **platforms:** directorio en el que se incluyen las instrucciones para compilar nuestra aplicación en las plataformas en las que se va a utilizar, así como el archivo correspondiente (apk o .ipa) para instalarla. El código correspondiente

a dichas instrucciones y la apk o .ipa se generan automáticamente al ejecutar la instrucción *ionic build*. Aparte de esto, también existe un fichero platforms.json en el que se indican todas las plataformas para las que se ha compilado la aplicación.

- **plugins:** en esta carpeta se encuentran los plugins y extensiones necesarios para dotar de funcionalidad a nuestra aplicación. En este proyecto tan solo están los plugins de Cordova e Ionic generados automáticamente pues no hemos añadido ninguno adicional.
- **resources:** este directorio contiene los recursos propios de cada plataforma, como por ejemplo el icono de la aplicación o el diseño de sus notificaciones. Al no haber diseñado el icono ni haber utilizado ningún recurso adicional, aquí aparecen todos aquellos que se generan por defecto al crear el proyecto.
- **src:** es la carpeta más importante de la aplicación en la que se incluye todo el código fuente. Por ello será analizada independientemente en detalle en el siguiente subapartado.
- **www:** en esta carpeta está la vista web de la aplicación. El contenido de esta carpeta se genera y actualiza automáticamente cada vez que cambiamos el código de *src* y lo compilamos. Aquí no se debe modificar nada ya que todo lo que hay es generado de forma automática y se renueva cada vez que se cambia algo en el código.

El resto de ficheros que podemos ver en el directorio son los siguientes:

- **.editorconfig** y el siguiente fichero que aparece sin nombre: son ficheros usados por el editor de texto y por GIT y no debemos tocarlos ni preocuparnos de ellos.
- **config.xml:** en este fichero se incluyen parámetros que se generan automáticamente al crear un proyecto con Ionic. Lo más importante es que aquí es donde se especifican los permisos que necesitará la aplicación en el dispositivo en el que vaya a ser instalada. Como nuestra aplicación no necesita ningún permiso extra este fichero no se ha modificado.
- **ionic.config.json:** contiene información básica sobre el proyecto y la configuración de Ionic utilizada en su desarrollo. También se genera automáticamente y no ha sido necesario modificarlo.
- **package.json:** fichero utilizado por NodeJS para compilar y gestionar el directorio *node_modules* y donde se listan todos los paquetes y dependencias que hay en el proyecto.
- **tsconfig.json y tslint.json:** ambos ficheros son similares y contienen información para compilar el código TypeScript. Se generan automáticamente y no es necesario modificarlos.

5.3 Código fuente: directorio src

Como se ha dicho anteriormente el directorio src es el más importante de nuestra aplicación y en él se encuentra el código fuente. Por ello vamos a analizarlo detalladamente por separado. En dicho directorio es donde se realiza fundamentalmente toda la parte de codificación del proyecto. A continuación se muestra una imagen correspondiente al contenido de dicho directorio:

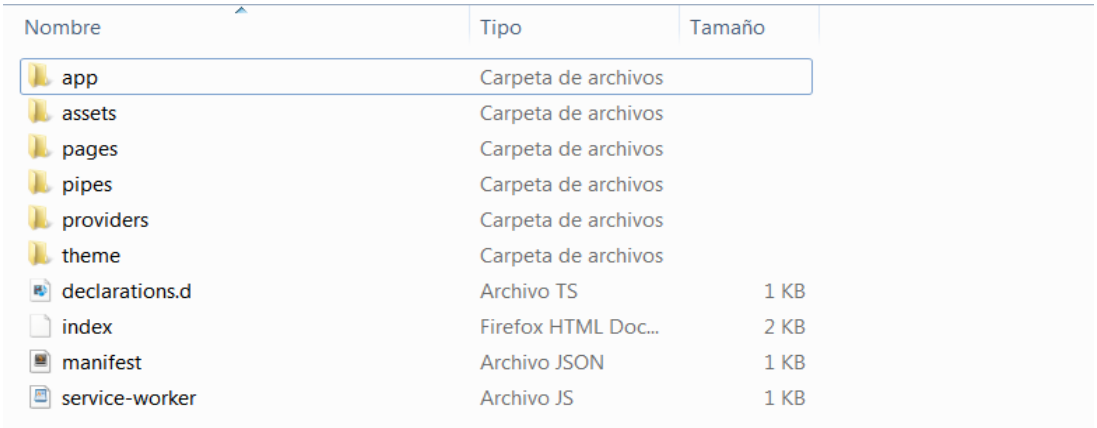

Figura 5-2: directorio src en donde se encuentra el código fuente

Al igual que hicimos con el directorio principal del proyecto, vamos a comentar detalladamente lo que hay en esta carpeta:

- **app:** esta es la carpeta donde se encuentra la raíz de la aplicación. Tiene 5 ficheros:
  - **app.html:** aquí se establece la pantalla raíz de la aplicación. En nuestro caso será la pantalla de login.
  - **app.component.ts:** declaración de la clase MyApp correspondiente a la aplicación e inicialización de la pila de navegación en la pantalla de inicio establecida en el fichero anterior.
  - **app.module.ts:** declaración de todos los módulos y componentes necesarios para la utilización de la aplicación e importación de los mismos dentro del proyecto. Cada vez que se crea una ventana nueva en nuestra aplicación o se añade una librería se debe actualizar este fichero declarando el nuevo elemento para que quede añadido a la aplicación.
  - **app.scss:** fichero global de estilos y diseño. Al no haber modificado nada del diseño de la aplicación este fichero está vacío.
  - **main.ts:** fichero donde se llama a la función AppModule que da inicio a la aplicación y hace que comience la ejecución.
- **assets:** en esta carpeta se incluyen las imágenes, GIFs y demás elementos que vayan a aparecer en la aplicación. Como en nuestra aplicación no se ha incluido ninguna imagen o similar este directorio se encuentra vacío.
- **pages:** carpeta donde se alojan todas las ventanas o vistas que tiene nuestra aplicación. Cada una de las ventanas dispone de un subdirectorio propio llamado con el nombre que le demos a dicha vista. Dentro de cada subdirectorio hay tres ficheros cuyo nombre también es igual al nombre de la vista a la que pertenecen:
  - **Fichero .ts:** código TypeScript correspondiente a la vista. En este fichero se incluyen todas las funciones necesarias para gestionar la lógica que haya en esa página. Además también se incluyen las funciones que conectan la vista con el *provider* (modelo) y viceversa en aquellos lugares en los que sea necesario.
  - **Fichero .html:** código HTML en donde están todos los elementos que aparecen en esa vista. Es equivalente al fichero HTML de cualquier página web.

    - **Fichero .scss:** fichero de diseño y estilos correspondiente a esa vista. Este fichero es el equivalente al fichero CSS en una página web. Aquí se establecen parámetros como los colores de fondo, el tamaño y estilo de la letra, los márgenes entre elementos, etc.
- **pipes:** aquí es donde irán los ficheros correspondientes a los *pipes*, es decir, aquellas partes de la aplicación encargadas de formatear o tratar los datos antes de ser mostrados. Estarían en un paso intermedio entre el *provider* y la vista. El único pipe que hay en nuestra aplicación se llama "Jugadores" y es el que se encarga de mostrar las alineaciones de ambos equipos en forma de tabla. Para ello obtiene del *provider* el listado de jugadores y lo pone en la forma adecuada para poder ser introducido en una tabla HTML en la vista.
- **providers:** como ya se comentó anteriormente, los providers en Ionic son servicios que equivalen al modelo y es en ellos donde se guardan los datos con los que trabaja la aplicación. En esta carpeta se incluyen los ficheros correspondientes a dichos servicios, teniendo dos:
    - **auth-service.ts:** servicio de inicio de sesión que permitirá realizar el login de usuarios conectándose a una base de datos cuando la tengamos implementada. Actualmente no se utiliza.
    - **data.ts:** servicio donde se guardan todos los datos necesarios en la aplicación en clases de JavaScript. Aquí es donde se guardan los nombres de los equipos, las alineaciones, las incidencias, los eventos y todo aquello que introduzca el usuario.
- **theme:** carpeta utilizada para ficheros relacionados con el diseño global de la aplicación. Contiene un archivo variables.scss generado por defecto en donde se pueden establecer los tipos y colores de letra, colores de fondo y demás parámetros que irían en un fichero CSS global en una página web. Este archivo no se ha modificado al no haber trabajado el diseño de la aplicación por lo cual está en el mismo estado que al generarla. Lo que se incluya en este fichero será utilizado por defecto en cada una de las vistas salvo que en el propio fichero .scss de la vista se especifique algo diferente.

El resto de ficheros que podemos ver en el directorio *src* son:

- **declarations.d.ts:** fichero auxiliar donde se pueden hacer declaraciones de elementos o librerías externos al proyecto que no estén en el fichero app.module.ts. En nuestro caso este fichero está vacío.
- **index.html:** fichero principal de la aplicación. En el *body* de este documento aparece la etiqueta <ion-app>, dentro de la cual se cargará el fichero app/app.html y con él toda la aplicación. Este fichero se genera automáticamente al hacer la compilación. En su *head* aparecen los *links* y *scripts* correspondientes a todas las librerías y ficheros de estilos que se van a utilizar en la aplicación, así como el título de la misma.
- **manifest.json:** fichero con información básica sobre las vistas y el proyecto. En él se contiene el nombre de la aplicación, el nombre de la vista inicial, la orientación inicial (en nuestro caso vertical), la ruta al icono de la aplicación y sus dimensiones y los colores por defecto de fuente y fondo.
- **service-worker.js:** en este fichero se establecen todos aquellos elementos que se deben guardar en la caché al ejecutar la aplicación. Este fichero se ha dejado por defecto y las cosas que se incluyen en él son: el fichero app/main.ts, el fichero index.html y el fichero manifest.json.

5.4 Implementación del modelo

En este subapartado vamos a comentar los detalles de la implementación del modelo. Como se ha comentado anteriormente, el modelo estará implementado en TypeScript (variante de JavaScript) y el fichero en el que se ha realizado está en la ruta src/providers/data.ts de nuestro proyecto.

Para la implementación del modelo hemos optado por utilizar las estructuras de JavaScript que mejor se ajustaban a los datos que necesitábamos guardar en cada caso. En resumen:

- **Partido:** hemos creado un Object de JavaScript como elemento principal de nuestra estructura de datos. Dicho Object contendrá: otros dos Object correspondientes a ambos equipos, dos elementos de tipo Date correspondientes a las horas de inicio de ambas partes y un array donde se guardarán las incidencias.
- **Equipos:** cada uno de los dos equipos será un Object. Dicho Object contendrá un String con el nombre del equipo, un HashMap de jugadores y otro de técnicos, un array de sustituciones, un contador de goles, un contador de jugadores titulares y otro contador de jugadores suplentes.
- **Alineaciones y técnicos:** para guardar las alineaciones y el listado de técnicos de ambos equipos hemos utilizado un HashMap. Esta estructura de datos aloja los elementos con pares clave-objeto y no permite que dentro de dicha estructura se repitan las claves. Esto ha sido lo más importante a la hora de decantarnos por esta estructura, ya que nos permitirá tener el control a la hora de evitar la repetición de dorsales en los jugadores y de puestos en los técnicos. Como es lógico, la clave escogida ha sido el dorsal en los jugadores y el puesto en los técnicos. El objeto son todos los datos correspondientes a cada uno de ellos. De esta forma se guardan estos datos, teniendo para cada equipo un HashMap de jugadores y otro de técnicos.
- **Incidencias:** para guardar las incidencias hemos utilizado un Object de JavaScript en el que hemos establecido un campo *tipo* y un campo *descripción*. Cada incidencia será un objeto y todas las correspondientes a un partido se guardarán en un array.
- **Horas de inicio de ambas partes:** se corresponden con un elemento de tipo Date.
- **Goles:** los goles se guardarán en un Object en donde habrá un campo *minuto* y un campo *tipo* (normal, de penalti o en propia puerta). Cada gol quedará asignado al jugador correspondiente, el cual tendrá un array de goles en donde se irán añadiendo a medida que se le asignan.
- **Tarjetas:** la gestión de las tarjetas es similar a la de los goles. Se guardarán en un Object en donde habrá un campo *minuto*, un campo *motivo* y un campo *color*. A la hora de asignarlas al jugador se meterán en un array de tarjetas y sobre dicho array se establecerá un control para saber cuándo a un jugador se le asigna una segunda tarjeta amarilla.
- **Sustituciones:** las sustituciones se guardarán en un array dentro de cada equipo. Cada sustitución tendrá un campo *titular* y otro *suplente* donde se guardarán los dorsales del jugador que entra y el que sale. También tendrá un campo *minuto* en el que se guardará el minuto en el que se realiza.

5.5 Implementación de las vistas

Como ya se mencionó anteriormente cada vista se compone de un fichero TypeScript, un fichero HTML y un fichero CSS. Como el estilo se ha dejado prácticamente por defecto, tan solo hemos trabajado con los dos primeros. A continuación se resumirán los aspectos más relevantes a tener en cuenta de la codificación de las vistas:

- **Ficheros TypeScript (.ts):** los ficheros TypeScript prácticamente no influyen en la vista, sino en su interacción con el modelo, por ello serán tratados en el siguiente subapartado.
- **Ficheros HTML:** aquí se han utilizado muchos de los elementos de los que nos provee Ionic. Estos ficheros están estructurados en dos partes. La primera parte está entre las etiquetas <ion-header> y ahí se introduce el título (<ion-title>) de la vista y los botones de la barra superior (botón para volver a la vista anterior y de logout en nuestra aplicación). La segunda parte está entre las etiquetas <ion-content> y ahí es donde está el contenido de la vista. En ese contenido se utilizan varios elementos de Ionic. Entre ellos están los botones (<ion-button>), pestañas (<ion-tabs>), formularios (<ion-form>), tablas (<ion-table>), objetos (<ion-item>), etc. Para poner estos elementos en nuestra vista basta con escribir las etiquetas HTML con los nombres correspondientes. El resto de la codificación de estos ficheros ha sido similar al que se haría en la programación de una página web.

5.6 Interacción modelo-vista

La interacción entre el modelo y la vista es una de las novedades de este proyecto ya que hace que el tradicional MVC sea sustituido por un MVVM. Como se comentó anteriormente esta comunicación entre el modelo y la vista se realiza por llamadas a funciones de JavaScript. La forma de realizar esto se describe a continuación:

- **Modificación o cambios en la vista:** en cada vista en la que se vayan a introducir datos o a realizar cambios en el modelo, se incluye dentro del código HTML los inputs correspondientes dentro de un formulario. Cada formulario tendrá un botón de envío de datos (submit) que al ser pulsado ejecutará una función de JavaScript. Dicha función analizará la entrada para ver si es correcta, dando un aviso si contiene algún dato inválido. Si la entrada es correcta, el modelo (*provider*) será actualizado en consecuencia.
- **Modificación o actualización del modelo:** si como consecuencia del flujo de la aplicación o de la introducción de datos por parte del usuario se producen cambios en el modelo, estos serán mostrados en la vista. A la hora de mostrar en la vista el estado del modelo se utiliza código JavaScript que se comunica con el *provider* y obtiene la información. Dicho código es ejecutado cada vez que se carga la vista, por tanto la información que está en el modelo se va recargando automáticamente. Al estar el modelo permanentemente actualizado, la información que se muestra en la vista también lo estará.

Podemos ver todo esto tomando como ejemplos la vista en la que se introducen los jugadores y la vista en la que se muestran las alineaciones:

- En la vista en la que se introducen los jugadores podemos ver como en su código HTML creamos un formulario (<ion-form>) con las entradas del usuario (<ion-input> o <ion-checkbox>) correspondientes a cada jugador. A su vez este formulario llamará a una función llamada *anhadirJugador()* cuando se pulse el botón de submit. Todo esto lo podemos ver en la figura 5-3.

```
<!--
  Generated template for the IntroducirJugadorLocal page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info on
  Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Introducir Jugador Local</ion-title>
  </ion-navbar>

</ion-header>


<ion-content padding>
  <form (ngSubmit)="anhadirJugador()">
    <ion-item>
      <ion-label color = "primary">Nombre:</ion-label>
      <ion-input placeholder="Text Input" id="nombreJugador"  [(ngModel)]="jugador.nombre" name="nombreJugador"></ion-input>
    </ion-item>
    <ion-item>
      <ion-label color = "primary">Dorsal:</ion-label>
      <ion-input type="number" placeholder="number" id="dorsalJugador"  [(ngModel)]="jugador.dorsal" name="dorsalJugador"></ion-input>
    </ion-item>
    <ion-item>
      <ion-label>Titular</ion-label>
      <ion-checkbox id="jugadorTitular"  [(ngModel)]="jugador.titular" name="jugadorTitular"></ion-checkbox>
    </ion-item>
    <ion-item>
      <ion-label>Capitan</ion-label>
      <ion-checkbox id="jugadorCapitan"  [(ngModel)]="jugador.capitan" name="jugadorCapitan"></ion-checkbox>
    </ion-item>
    <ion-item>
      <ion-label>Portero</ion-label>
      <ion-checkbox id="jugadorPortero"  [(ngModel)]="jugador.portero" name="jugadorPortero"></ion-checkbox>
    </ion-item>

    <button ion-button type="submit" color="secondary">Confirmar</button>
  </form>
</ion-content>

```

Figura 5-3: código HTML correspondiente a la vista de introducir jugador

- En el código JavaScript correspondiente a esa vista podemos ver cómo está definida la función *anhadirJugador()* a la que se llamará cuando el usuario pulse el botón "Confirmar" del formulario. Vemos como en dicha función se van leyendo las entradas que se introdujeron, accediendo a ellas por el valor de *ngModel* que se les dio en el código HTML. Previamente en dicha función se hace una comprobación de que las entradas sean correctas y válidas, mostrando los *alerts* correspondientes con el mensaje de error adecuado en el caso de que no lo sean. Una vez que se ha comprobado que las entradas son correctas, se actualiza el modelo, en nuestro caso el *provider* cuyo nombre es *data* y al cual accedemos con *this.data*. Podemos ver el código completo de dicha función en la figura 5-4.

```
public anhadirJugador() {
  if (this.jugador["dorsal"] <= 0 || this.jugador["dorsal"] >= 100){
    let alert = this.alertCtrl.create({
        title: 'Error',
        subTitle: 'Es obligatorio introducir un dorsal entre el 1 y el 99'
      });
      alert.present();
  }
  else if (this.jugador["nombre"]  == ""){
    let alert = this.alertCtrl.create({
        title: 'Error',
        subTitle: 'Es obligatorio introducir el nombre'
      });
      alert.present();
  }
  else if (this.jugador["titular"]  == false && this.jugador["capitan"]  == true){
    let alert = this.alertCtrl.create({
        title: 'Error',
        subTitle: 'El capitán debe ser un jugador titular'
      });
      alert.present();
  }
  else if (this.data.equipoLocal.get(this.jugador["dorsal"]) != null){
      let alert = this.alertCtrl.create({
        title: 'Error',
        subTitle: 'No puede haber dos jugadores con el mismo dorsal'
      });
      alert.present();
  }else{
    this.data.equipoLocal.set(this.jugador["dorsal"], this.jugador);
    this.nav.push(AlineacionLocalPage);
  }
}
```

Figura 5-4: código correspondiente a la función anhadirJugador()

- En el código HTML correspondiente a la vista en la que se muestra la alineación local podemos ver cómo se actualiza la vista al haber un cambio en el modelo. Esto sucede porque el modelo está permanentemente actualizado y cada vez que la vista se carga, se carga también su estado. Esto se realiza en nuestro ejemplo al realizar un *ngFor* sobre la lista de jugadores para mostrarlos todos. Dicho bucle de Ionic utiliza el *pipe jugadores* para mostrar la alineación entera. El *pipe* que tenemos es lo más parecido que puede haber a un controlador en este tipo de proyectos, pues actúa en una parte intermedia entre el *provider* y la vista y en este caso trata los datos que están en el modelo para que la vista los pueda introducir en una tabla HTML. Podemos ver el código HTML de la vista de mostrar la alineación en la figura 5-5 y el código JavaScript correspondiente al *pipe jugadores* en la figura 5-6.

```
<!--
  Generated template for the Alineacion page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info on
  Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Alineacion Local</ion-title>
  </ion-navbar>

</ion-header>


<ion-content padding>
  <h1 class="title">Nombre Equipo</h1>
    <div class="row header">
      <div class="col">Dorsal</div>
      <div class="col">Nombre</div>
      <div class="col">Goles</div>
      <div class="col">Tarjetas</div>
      <div class="col">Sustitución</div>
    </div>
    <div class="row" *ngFor="let entry of this.data.equipoLocal | jugadores"> <!--Se lee la alineacion -->
      <div class="col">{{entry.key}}</div>
      <div class="col">{{entry.value.nombre}}</div>
      <div class="col">{{entry.value.goles.length}}</div>
      <div class="col">{{entry.value.tarjetas}}</div>
      <div class="col">{{entry.value.sustituciones}}</div>
    </div>
<button ion-button (click)="verJugador()">Introducir Jugador</button>
</ion-content>
```

Figura 5-5: código HTML correspondiente a la vista de mostrar la alineación local

```
import { Pipe, PipeTransform } from '@angular/core';

/*
  Generated class for the Jugadores pipe.

  See https://angular.io/docs/ts/latest/guide/pipes.html for more info on
  Angular 2 Pipes.
*/

@Pipe({
    name: 'jugadores'
})
export class Jugadores implements PipeTransform {
 transform(value: any, args?: any[]): Object[] {
        let returnArray = [];

        value.forEach((entryVal, entryKey) => {
            returnArray.push({
                key: entryKey,
                value: entryVal
            });
        });

        return returnArray;
    }
}
```

Figura 5-6: código JavaScript correspondiente al pipe jugadores

6 Pruebas y resultados

6.1 Introducción

La fase de pruebas se ha realizado para encontrar errores cometidos durante el desarrollo, posibles mejoras de usabilidad o la inclusión de nuevas funcionalidades. Como se ha comentado anteriormente, se estableció un desarrollo ágil entre la fase de desarrollo y la fase de pruebas, así que al finalizar un periodo de pruebas se volvía a codificar para cambiar todos aquellos aspectos susceptibles de mejora. En este capítulo se realiza una descripción de las pruebas realizadas y los resultados obtenidos.

6.2 Pruebas

Para realizar las pruebas se han utilizado los navegadores web, emuladores y dispositivos físicos comentados en el capítulo 3 de este documento. La ventaja de realizar las pruebas en navegadores web es que las puedes realizar en el mismo equipo en el que estás haciendo la codificación, siendo un proceso muy rápido el de modificar el código y pudiendo hacerlo a la vez que haces las pruebas. Lo mismo sucede con los emuladores, que al ser ejecutados en el mismo equipo te permiten realizar modificaciones de forma rápida a la vez que observas cómo se vería la aplicación en el dispositivo que estén emulando.

Dentro de las pruebas realizadas podemos diferenciar tres tipos: las pruebas unitarias en las que se prueba cada módulo por separado; las pruebas de compatibilidad en las que se prueba la aplicación con diferentes sistemas operativos y tamaños de pantalla; y las pruebas de validación y verificación donde se prueba el *prototipo* o producto final en su ámbito de uso.

6.2.1 Pruebas unitarias

En este tipo de pruebas se evaluaron los módulos y clases que tenía nuestra aplicación. El objetivo era buscar errores cometidos en la fase de desarrollo y darles solución. Todas estas pruebas se realizaron en navegador web para poder corregir los errores más fácilmente. Entre las pruebas realizadas están las siguientes:

- El usuario puede crear un partido correctamente introduciendo los nombres de ambos equipos y su categoría.
- El usuario puede seleccionar una hora de inicio del partido y de inicio de la segunda parte.
- El usuario puede introducir jugadores a la alineación de un equipo.
- Si una alineación no tiene el número de jugadores correcto, se muestra el aviso correspondiente.
- Si una alineación no tiene capitán o portero, se muestra el aviso correspondiente.
- Si un jugador se introduce incorrectamente, se muestra el aviso correspondiente.
- El usuario puede introducir técnicos a un equipo.
- El usuario puede asignar un gol a un jugador titular. Si el gol no se asigna correctamente, se muestra el aviso correspondiente.
- El usuario puede asignar una tarjeta a cualquier participante del partido. Si la tarjeta no se asigna correctamente, se muestra el aviso correspondiente.
- Si es asignada una segunda tarjeta amarilla a un participante, la aplicación muestra el aviso correspondiente.

- Si un jugador es expulsado, no se le pueden asignar goles ni sustituciones desde ese momento.
- El usuario puede añadir la sustitución de un jugador titular por un jugador suplente.
- El usuario puede añadir una incidencia.
- Al finalizar el partido se genera un informe en PDF.

6.2.2 Pruebas de compatibilidad

Al haber realizado una aplicación multiplataforma, en este tipo de pruebas se intentó comprobar cómo funciona la aplicación con diferentes sistemas operativos. Al disponer únicamente de dispositivos físicos Android, las pruebas correspondientes a otros sistemas operativos se realizaron con emuladores. Para probar la aplicación con emuladores iOS basta con ejecutar *ionic run ios* en la consola de Ionic y la propia herramienta nos muestra nuestra aplicación ejecutada en un emulador correspondiente a un iPhone 6 Plus. Para Windows Phone hemos utilizado el emulador que se ofrece en la web de Microsoft [50] y para Blackberry el que se ofrece en la web de desarrolladores de dicha plataforma [51]. Para utilizar los emuladores basta con descargarlos e instalarlos, ponerlos en ejecución e instalar en ellos la apk generada.

También hemos hecho pruebas de compatibilidad con diferentes tamaños de pantalla. Para ello hemos utilizado los dispositivos físicos Android comentados en el capítulo 3 y comprobamos que la aplicación se mostraba correctamente en ambos.

6.2.3 Pruebas de validación y verificación

Este tipo de pruebas se realizaron en partidos reales con el objetivo principal de evaluar la usabilidad de la aplicación y explorar nuevas funcionalidades. Al estar realizadas previamente las pruebas unitarias, en esta nueva fase de pruebas el número de errores de programación no resueltos era mínimo. Para realizarlas se utilizaron dispositivos físicos Android en los que se instaló la apk siguiendo el manual que se puede ver en el Anexo A de este documento.

Estas pruebas se hicieron en dos etapas. En la primera de ellas se probó la aplicación en partidos que estaban siendo visualizados por televisión para evitar que una falta de experiencia en su utilización o posibles errores influyesen o ralentizasen un partido real. Una vez se superó con éxito esta fase, se procedió a utilizar la aplicación en el arbitraje de partidos reales a la vez que se realizaban las anotaciones manuales.

6.3 Resultados

El resultado final de este Trabajo de Fin de Grado es una aplicación móvil que permite sustituir las anotaciones manuales que se realizan en un partido de fútbol. Tras sucesivas pruebas y varios prototipos se llegó a una versión final que se puso en funcionamiento.

Los resultados de las pruebas de validación y verificación fueron satisfactorios ya que la aplicación cumplió todos los objetivos previstos en un partido real. Tras haber superado con éxito dichas pruebas se considera finalizada la aplicación y se pondrá en marcha y se utilizará en partidos reales.

En el anexo D se muestran las capturas de pantalla realizadas en dispositivo físico correspondientes a las pantallas más relevantes de la aplicación.

7 Conclusiones y trabajo futuro

7.1 Conclusiones

Tras la realización de este Trabajo de Fin de Grado se ha conseguido desarrollar una aplicación móvil que sustituye las anotaciones manuales en un partido de fútbol. Además se ha utilizado por primera vez el framework Ionic para construir aplicaciones móviles multiplataforma.

Se considera que la realización del proyecto ha sido satisfactoria ya que se han cumplido los objetivos planteados inicialmente. La mayor dificultad a la hora de llevar a cabo su realización ha sido familiarizarse con el nuevo entorno, la nueva arquitectura MVVM y la estructura novedosa del proyecto con la que no se había trabajado nunca anteriormente. Además del éxito en el cumplimiento de los objetivos planteados también es satisfactorio el hecho de haber aprendido a utilizar una nueva herramienta que parece bastante interesante.

Durante la realización de este Trabajo de Fin de Grado se han aplicado los conocimientos aprendidos en varias asignaturas del Grado en Ingeniería Informática. En concreto se han utilizado los conocimientos sobre programación web aprendidos en *Sistemas Informáticos I* (utilización de lenguajes HTML, CSS, JavaScript); los conocimientos relacionados con el análisis y diseño de proyectos y la educción de requisitos aprendidos en las asignaturas de *Proyecto de Análisis y Diseño de Software*, *Ingeniería del Software* y *Proyecto de Ingeniería del Software*; y los conocimientos sobre el diseño y el flujo de una aplicación móvil aprendidos en la asignatura de *Desarrollo de Aplicaciones para Dispositivos Móviles.* El haber podido emplear los conocimientos adquiridos a lo largo de la carrera ha hecho que la experiencia de la elaboración de este Trabajo de Fin de Grado haya sido aún más satisfactoria y ha permitido afianzar y reforzar dichos conocimientos.

En el siguiente subapartado se expondrán posibles mejoras a realizar sobre la aplicación, ya que en esta primera versión se ha buscado lograr la funcionalidad deseada y no se ha hecho tanto énfasis en algunos aspectos como el diseño gráfico. Además se espera que con la utilización de la aplicación por parte de varios usuarios se vayan recibiendo sugerencias de mejora o de nuevas funcionalidades a añadir.

7.2 Trabajo futuro

Los objetivos que se pretendían alcanzar con esta aplicación se pueden considerar cumplidos con éxito, no obstante todavía hay ciertos elementos susceptibles de mejora en un trabajo futuro. Vamos a tratarlos a continuación:

- **Posibilidad de ajustes de idioma:** en un principio se ha programado la aplicación en lengua castellana, pues ha sido pensada para ser para ser utilizada en partidos de ligas de la federación española. Si el proyecto se extendiese, sería recomendable añadirle a la aplicación la posibilidad de ajustar su idioma y que pudiese ser mostrada en otras lenguas, como por ejemplo en inglés.
- **Implementación de un sistema de login y registro de usuarios:** actualmente la aplicación tiene una pantalla de login pero el acceso es libre ya que no tenemos usuarios registrados ni base de datos. La pantalla de login se ha implementado por si en el futuro aumenta el número de usuarios de la aplicación y se necesitase realizar una gestión de quienes la usan.

- **Mejora en el diseño gráfico:** El diseño gráfico podría mejorarse haciendo que la aplicación resultase estética además de funcional, siendo así más llamativa. Además, se podría añadir un icono a la aplicación y un diseño propio para sus notificaciones y no utilizar los que ofrece Ionic por defecto.
- **Inclusión de un servidor que permita mostrar resultados en tiempo real:** otra de las posibles mejoras que podría adoptarse, sería el despliegue de un servidor al cual la aplicación se conectase para enviarle los datos. Esto permitiría que el servidor mostrase en tiempo real el estado y la situación del partido.
- **Posibilidad de cargar las alineaciones online y no tener que introducir manualmente a los jugadores:** actualmente en la aplicación se introducen los nombres y dorsales de los jugadores manualmente antes del partido. Una posible mejora sería que dichos nombres pudiesen ser descargados de la base de datos de licencias de la Federación competente cuando éstos estuviesen disponibles.
- **Extensión a otros deportes:** la aplicación ha sido pensada para ser utilizada en partidos de fútbol. Una posible mejora de la misma podría ser su adaptación a otros deportes.
- **Introducción de datos por voz:** actualmente la interacción del usuario con la aplicación es a través de la pantalla táctil. Si las entradas del usuario se pudiesen hacer por voz mejoraría mucho la usabilidad de la aplicación.

Referencias

[1] *Ionic Framework:* https://ionicframework.com/ (Consultado por última vez el 30/05/2017)
[2] *Google Maps:* https://play.google.com/store/apps/details?id=com.google.android.apps.maps&hl=es (Consultado por última vez el 30/05/2017)
[3] *PassWallet:* https://play.google.com/store/apps/details?id=com.attidomobile.passwallet&hl=es (Consultado por última vez el 30/05/2017)
[4] *PassBook:* https://es.wikipedia.org/wiki/Passbook (Consultado por última vez el 30/05/2017)
[5] *Apple Pay:* https://www.apple.com/es/apple-pay/ (Consultado por última vez el 30/05/2017)
[6] *Bizum:* "Bizum, la plataforma de pago por móvil de los bancos, supera los 300000 usuarios", ABC del 10/01/2017. Consultado por última vez el 30/05/2017: http://www.abc.es/tecnologia/moviles/telefonia/abci-bizum-plataforma-pago-movil-bancos-supera-300000-usuarios-201701101618_noticia.html
[7] *Google Play:* https://es.wikipedia.org/wiki/Google_Play (Consultado por última vez el 30/05/2017)
[8] *App Store:* https://es.wikipedia.org/wiki/App_Store (Consultado por última vez el 30/05/2017)
[9] http://www.pcworld.com/article/161410/article.html (Consultado por última vez el 30/05/2017)
[10] https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store (Consultado por última vez el 30/05/2017)
[11] https://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store/ (Consultado por última vez el 30/05/2017)
[12] https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ (Consultado por última vez el 30/05/2017)
[13] *Runtastic GPS Running Fitness:* https://play.google.com/store/apps/details?id=com.runtastic.android (Consultado por última vez el 30/05/2017)
[14] *Nike + Run Club:* https://play.google.com/store/apps/details?id=com.nike.plusgps&hl=es_419 (Consultado por última vez el 30/05/2017)
[15] *Google Fit:* https://play.google.com/store/apps/details?id=com.google.android.apps.fitness&hl=es_419 (Consultado por última vez el 30/05/2017)
[16] *Golf Scorecard Pro:* https://play.google.com/store/apps/details?id=com.acteon.sports.golfscorecardpro&hl=es_419 (Consultado por última vez el 30/05/2017)
[17] *Estadísticas de baloncesto Omnistats:* *https://play.google.com/store/apps/details?id=com.aortizga.soft.basketball&hl=es_419* (Consultado por última vez el 30/05/2017)
[18] *Tracking Tennis Matches:* https://play.google.com/store/apps/details?id=com.trackingtennismatches&hl=es_419 (Consultado por última vez el 30/05/2017)
[19] *Marcador de fútbol electrónico:* *https://play.google.com/store/apps/details?id=com.chapas.cyclingsimulator&hl=es* (Consultado por última vez el 30/05/2017)
[20] *Referee Pro: https://play.google.com/store/apps/details?id=com.siine.referee* (Consultado por última vez el 30/05/2017)
[21] *Árbitro de bolsillo: https://play.google.com/store/apps/details?id=pocket.referee* (Consultado por última vez el 30/05/2017)
[22] *Cuarto Árbitro:* https://play.google.com/store/apps/details?id=com.simonmartin.arbitro (Consultado por última vez el 30/05/2017)
[23] *iReferee: https://play.google.com/store/apps/details?id=co.tapp.refereedroid* (Consultado por última vez el 30/05/2017)
[24] *Desarrollo en cascada: https://es.wikipedia.org/wiki/Desarrollo_en_cascada* (Consultado por última vez el 30/05/2017)

[25] *Desarrollo ágil:* https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software (Consultado por última vez el 30/05/2017)
[26] *Android Studio:* https://developer.android.com/studio/index.html (Consultado por última vez el 30/05/2017)
[27] *Android:* https://www.android.com/intl/es_es/ (Consultado por última vez el 30/05/2017)
[28] *iOS:* https://support.apple.com/es-es/ios (Consultado por última vez el 30/05/2017)
[29] *Blackberry:* https://global.blackberry.com/es/index (Consultado por última vez el 30/05/2017)
[30] *Windows Phone:* http://windowsphone.com/ (Consultado por última vez el 30/05/2017)
[31] *JavaScript:* https://www.javascript.com/ (Consultado por última vez el 30/05/2017)
[32] *TypeScript:* https://www.typescriptlang.org/ (Consultado por última vez el 30/05/2017)
[33] *HTML:* https://es.wikipedia.org/wiki/HTML (Consultado por última vez el 30/05/2017)
[34] *CSS:* https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada (Consultado por última vez el 30/05/2017)
[35] *Sublime Text:* https://www.sublimetext.com/ (Consultado por última vez el 30/05/2017)
[36] *Mocking Bot:* https://mockingbot.com/ (Consultado por última vez el 30/05/2017)
[37] *GIT:* https://git-scm.com (Consultado por última vez el 30/05/2017)
[38] *Bitbucket:* https://bitbucket.org (Consultado por última vez el 30/05/2017)
[39] *Internet Explorer:* https://www.microsoft.com/es-es/download/internet-explorer.aspx (Consultado por última vez el 30/05/2017)
[40] *Mozilla Firefox: https://www.mozilla.org/es-ES/firefox/new/* (Consultado por última vez el 30/05/2017)
[41] *Google Chrome:* https://www.google.es/chrome/browser/desktop/index.html (Consultado por última vez el 30/05/2017)
[42] *Apk:* https://es.wikipedia.org/wiki/APK_(formato) (Consultado por última vez el 30/05/2017)
[43] *jsPdf:* https://github.com/MrRio/jsPDF (Consultado por última vez el 30/05/2017)
[44] *AngularJS:* https://angularjs.org/ (Consultado por última vez el 30/05/2017)
[45] *NodeJS:* https://nodejs.org/es/ (Consultado por última vez el 30/05/2017)
[46] *Cordova:* https://cordova.apache.org/ (Consultado por última vez el 30/05/2017)
[47] *MVC:* https://en.wikipedia.org/wiki/Model-view-controller (Consultado por última vez el 30/05/2017)
[48] MVVM: https://en.wikipedia.org/wiki/Model-view-viewmodel (Consultado por última vez el 30/05/2017)
[49] *Provider:* https://docs.angularjs.org/guide/providers (Consultado por última vez el 30/05/2017)
[50] *Emulador Windows Phone:* https://msdn.microsoft.com/es-es/library/windows/apps/ff402563(v=vs.105).aspx (Consultado por última vez el 30/05/2017)
[51] *Emulador Blackberry:* http://developer.blackberry.com/develop/simulator/simulator_installing.html (Consultado por última vez el 30/05/2017)
[52] *Tutorial de inicio con Ionic:* http://ionicframework.com/getting-started/ (Consultado por última vez el 30/05/2017)

Glosario

Alert	Ventana emergente que aparece en las aplicaciones para mostrarle avisos al usuario relacionados con su actividad o el estado de la aplicación.
.apk	Extensión de los archivos correspondientes a aplicaciones para dispositivos Android. Se les suele llamar directamente "apk" y su nombre completo sería de Android Application Package.
Array	Un tipo de lista que permite guardar objetos y tenerlos ordenados, ofreciendo en ocasiones funciones de búsqueda, adicción y borrado de los elementos que contiene.
Framework	Representa una arquitectura de software con un conjunto de herramientas y métodos que sirven como referencia para realizar un posterior desarrollo.
HashMap	Tipo de estructura de datos que guarda los objetos en forma clave-valor. El valor es el propio contenido del objeto y la clave es un identificador del mismo. Dentro de la estructura no puede haber dos objetos con la misma clave.
.ipa	Extensión de los archivos correspondientes a aplicaciones para dispositivos iOS.
MVC	Siglas de Modelo-Vista-Controlador, un patrón de arquitectura de software.
MVVM	Siglas de Modelo-Vista-VistaModelo, un patrón de arquitectura similar al MVC.
Object (JavaScript)	Estructura genérica de datos que puede ser estructurada a gusto del programador haciendo que contenga todos los elementos y valores que desee.
Pipe	Elemento intermedio de Ionic entre el modelo y la vista que se encarga de formatear y tratar los datos antes de ser mostrados al usuario.
Prototipo	Producto que se obtiene al finalizar una iteración de ciclo de vida de un desarrollo ágil y que está preparado para ser utilizado a pesar de no tener todavía una funcionalidad completa.
Provider	Servicios utilizados por AngularJS que permiten dotar a una aplicación de alguna funcionalidad, normalmente relacionada con conexión a un servidor o manejo de datos e información.
Smartphone	Tipo de teléfono móvil inteligente que permite realizar numerosas actividades y que puede tener un comportamiento similar a una computadora.
Tablet	Dispositivo electrónico que tiene un tamaño intermedio entre un ordenador y un teléfono móvil.

Anexos

A Manual de instalación de la aplicación en un dispositivo Android

Para poder realizar todo el desarrollo de la aplicación, se supone que tenemos instalados en nuestro ordenador: NodeJS, Cordova, AngularJS e Ionic2. Todo ello se puede instalar siguiendo los pasos que aparecen en el tutorial oficial de Ionic: http://ionicframework.com/getting-started/

Para realizar la instalación de la aplicación en nuestro dispositivo móvil o tablet debemos seguir los siguientes pasos:

1. En la carpeta en la que tenemos nuestra aplicación debemos ejecutar el comando: *ionic build Android*

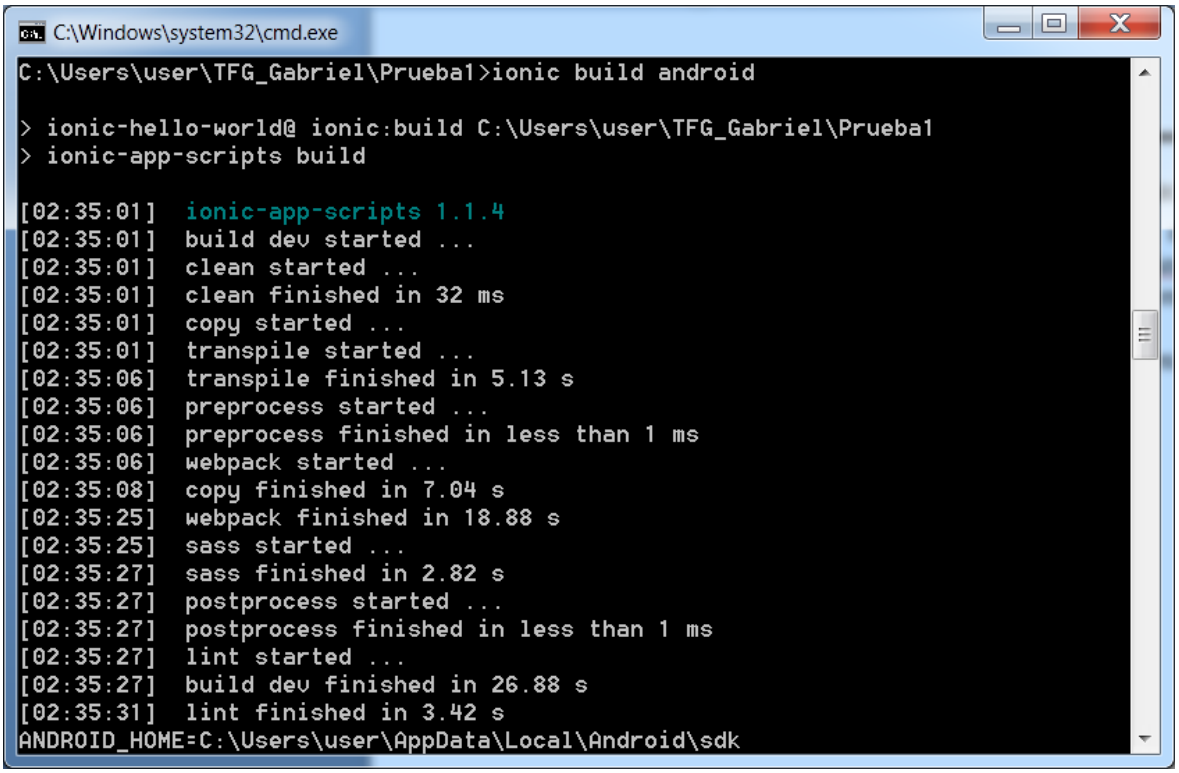

Figura A-1: generación de la apk de la aplicación

2. Debemos ir a la ubicación que nos indique la salida de la ejecución del comando y buscar el apk:

Figura A-2: ubicación en el ordenador del apk generado

3. Dicho apk deberá ser copiado o movido al dispositivo móvil mediante el medio correspondiente (cable, bluetooth, envío por email, etc). Una vez sea copiado, debemos buscarlo en la ubicación correspondiente en nuestro smartphone:

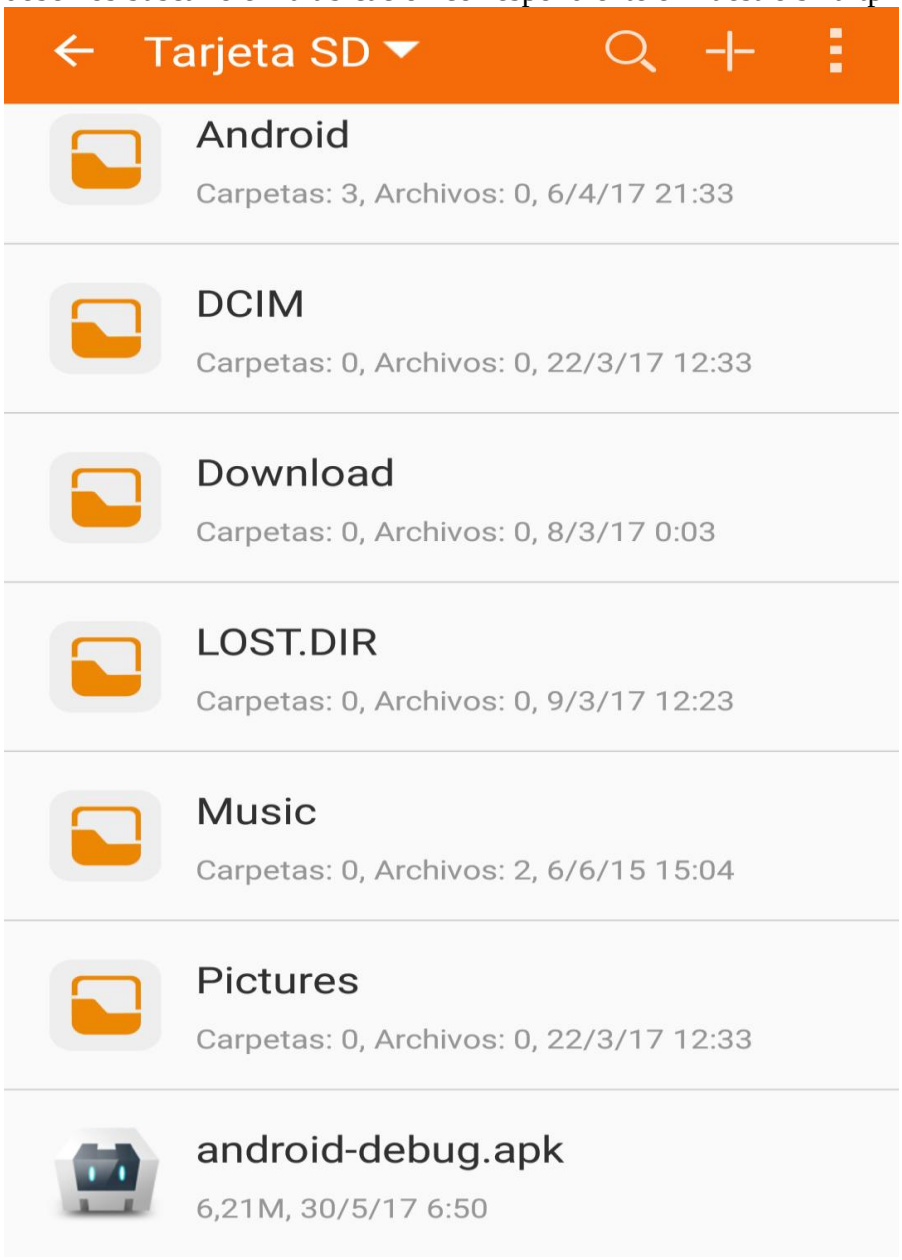

Figura A-3: ubicación en el dispositivo móvil del apk de la aplicación

Clicamos sobre él y lo instalamos, dándole a “Instalar” en el menú que nos aparezca:

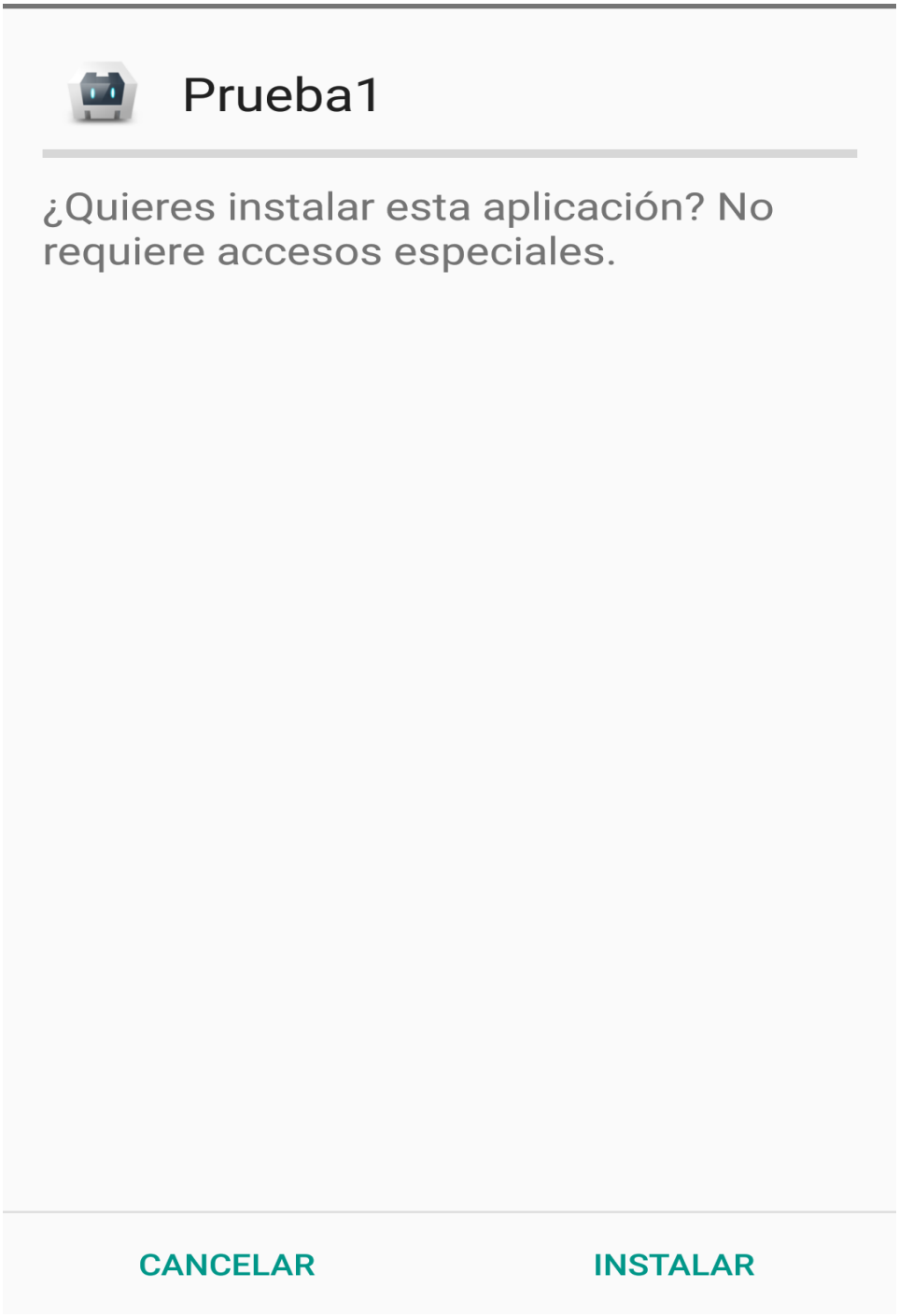

Figura A-4: aceptación de la instalación de la aplicación en el dispositivo móvil

4. Esperamos a que el sistema operativo instale la aplicación:

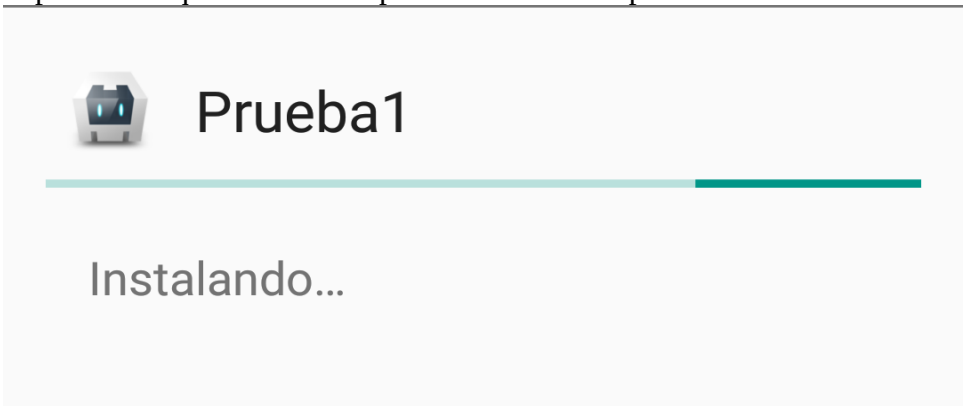

Figura A-5: instalación de la aplicación en el dispositivo móvil

5. Tras haber completado la instalación, la aplicación estará disponible y lista para ser abierta y utilizada.

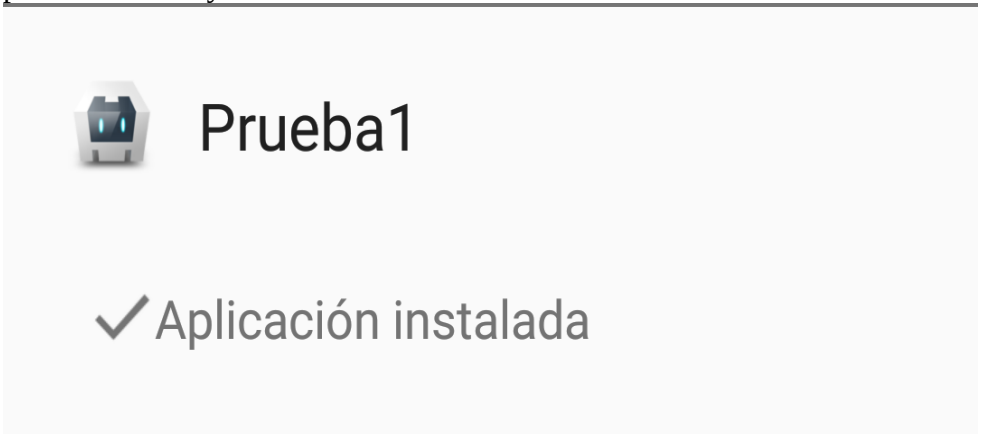

Figura A-6: aplicación instalada en el dispositivo móvil

La instalación de la aplicación en un dispositivo iOS es similar, sustituyendo el comando del paso 1 por *ionic build ios*. En lugar de generarse una apk se generará un archivo .ipa.

La instalación de la aplicación en dispositivos Windows Phone y Blackberry se podrá hacer utilizando la apk generada, pues dichos dispositivos permiten la instalación de aplicaciones Android.

B Manual del programador

En este apartado de la memoria vamos a comentar cómo instalar Ionic y cómo utilizarlo para crear aplicaciones. Un tutorial básico sobre como instalar Ionic se puede seguir en su página de tutoriales [52].

El primer paso a seguir es descargar NodeJS de su página oficial [45]. Una vez tenemos NodeJS podremos utilizarlo para instalar Cordova e Ionic. Para ello simplemente tendremos que escribir en una terminal la instrucción: *npm install -g cordova ionic*

Tras tener instalado Ionic, podremos crear una primera aplicación móvil. Para ello debemos situarnos en el directorio donde queremos crearla y abrir una terminal en él. En dicha terminal deberemos ejecutar la instrucción: *ionic start Name blank* donde "Name" será el nombre que queramos darle a nuestra aplicación. Sustituyendo "blank" por "tabs" o "sidemenu" se nos generarán aplicaciones con pestañas o con un menú lateral respectivamente.

Para crear nuevas páginas, *providers* o *pipes* en nuestra aplicación, debemos abrir una terminal en el directorio raíz y utilizar el comando *generate* de Ionic. Por ejemplo para crear una nueva página que se llame "Jugadores" deberemos ejecutar *ionic generate page Jugadores* y para crear el *provider* "Data" de nuestra aplicación tuvimos que ejecutar *ionic generate provider Data*.

Todas estas instrucciones crean los correspondientes directorios y generan código automáticamente. Nosotros durante el desarrollo simplemente debemos cambiar y programar aquellos ficheros susceptibles de modificación como se comentó en el capítulo 5.

Para probar una aplicación en el navegador web se debe abrir una terminal en el directorio raíz de la aplicación y ejecutar la instrucción *ionic serve*. Esta instrucción hace que la aplicación comience a servirse en una determinada dirección y puerto. De esta forma podremos probarla en el navegador. Lo mejor de esta instrucción es que cada vez que realizas una modificación en alguno de los ficheros de la aplicación y lo guardas, recompila automáticamente todo el proyecto y lo muestra automáticamente con los cambios realizados. En la figura B-1 podremos ver el resultado en la terminal tras ejecutar esta instrucción.

Para generar la apk o la .ipa de una aplicación se debe ejecutar en el directorio raíz las instrucciones *ionic build android* o *ionic build ios* respectivamente.

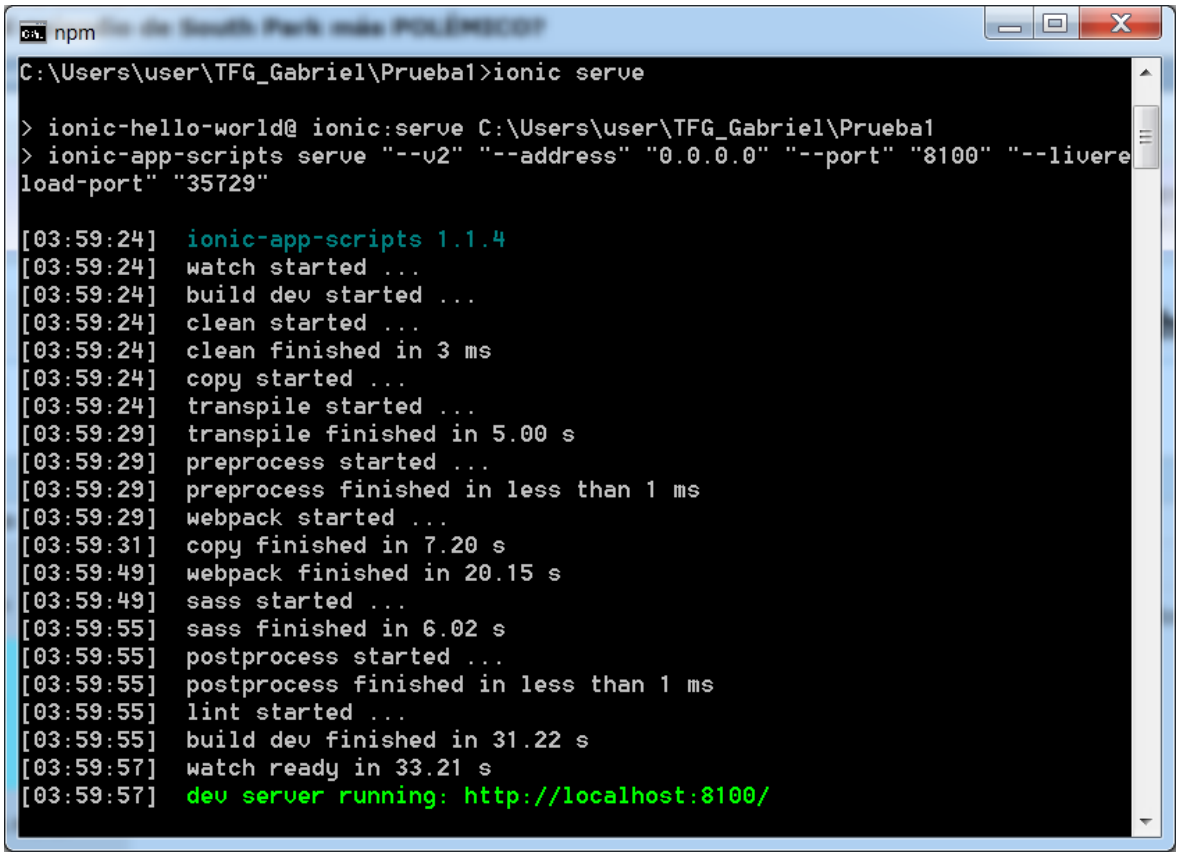

Figura B-1: ejecución de la instrucción ionic serve

C Maquetas de la interfaz gráfica

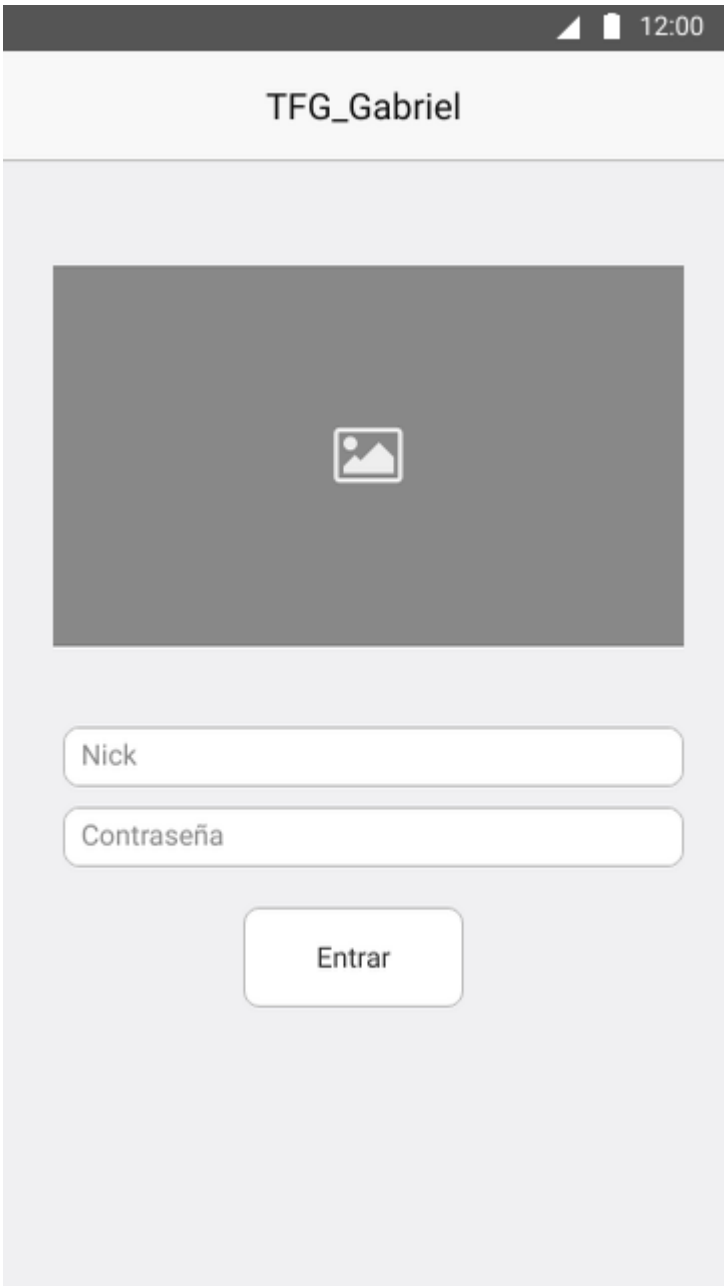

Figura C-1: maqueta correspondiente a la pantalla de inicio de la aplicación

Figura C-2: maqueta correspondiente a la pantalla de creación de nuevo partido

Figura C-3: maqueta correspondiente a la ventana de información general

Figura C-4: maqueta correspondiente a la ventana de información del equipo local

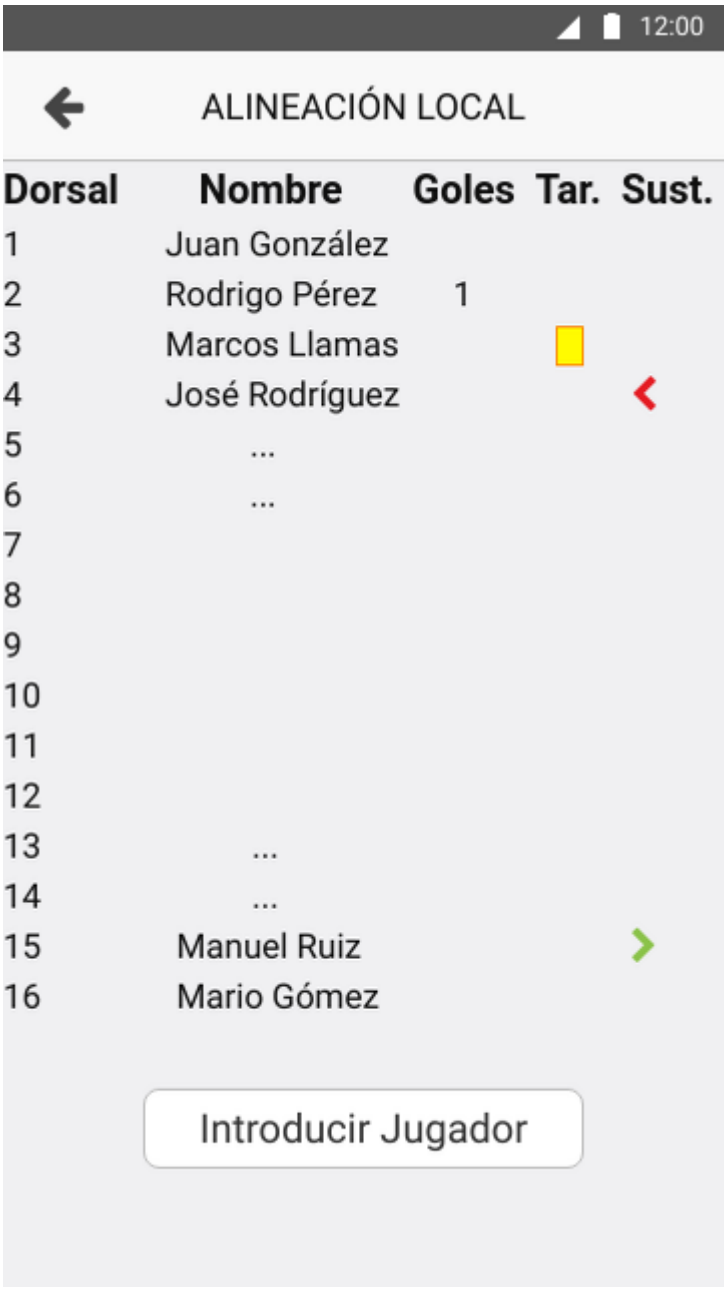

Figura C-5: maqueta correspondiente a la alineación del equipo local

Figura C-6: maqueta correspondiente a los técnicos del equipo local

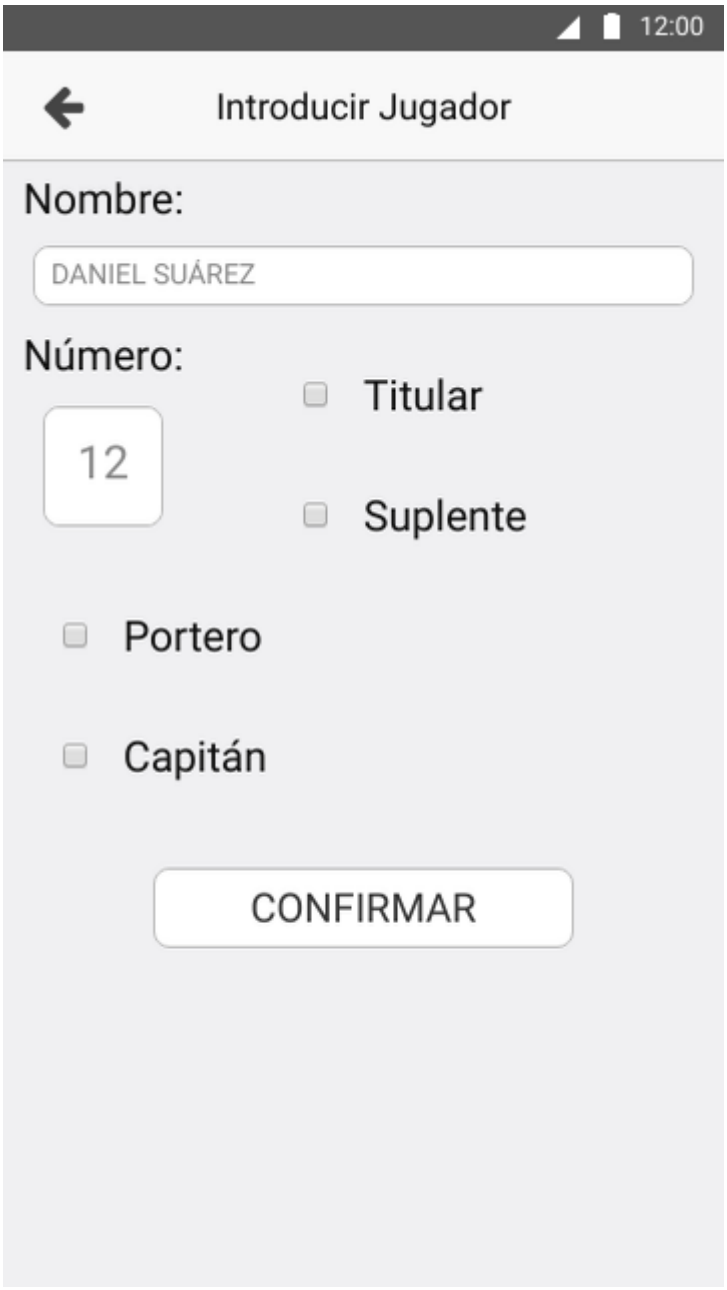

Figura C-7: maqueta correspondiente a la ventana de introducción de jugadores

Figura C-8: maqueta correspondiente a la ventana de introducción de técnicos

Figura C-9: maqueta correspondiente a la ventana de introducción de tarjetas

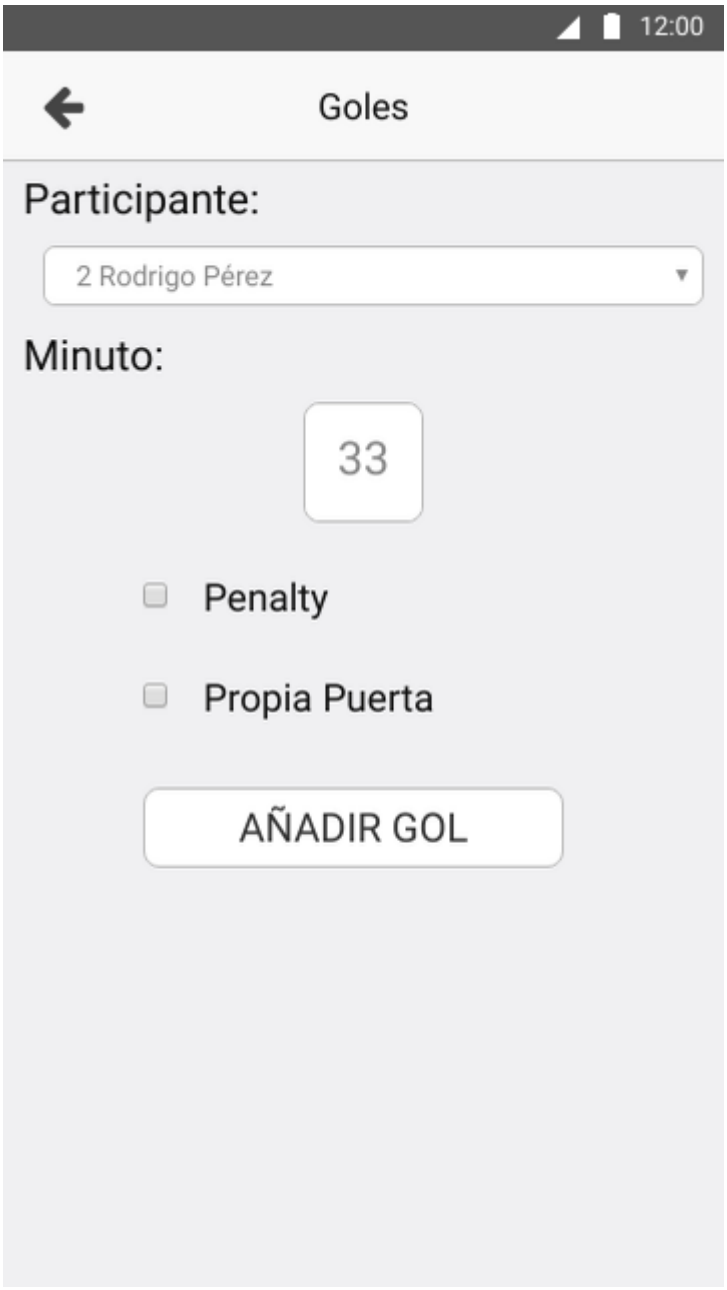

Figura C-10: maqueta correspondiente a la ventana de introducción de goles

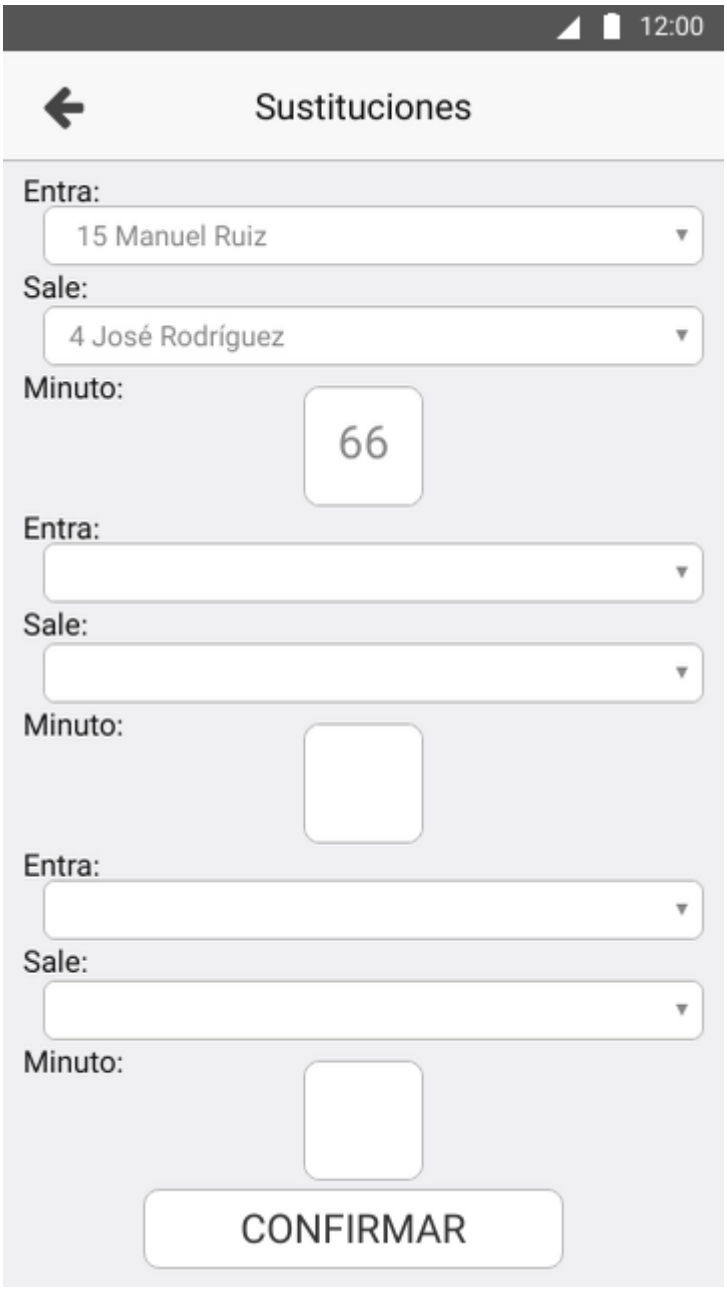

Figura C-11: maqueta correspondiente a la ventana de introducción de sustituciones

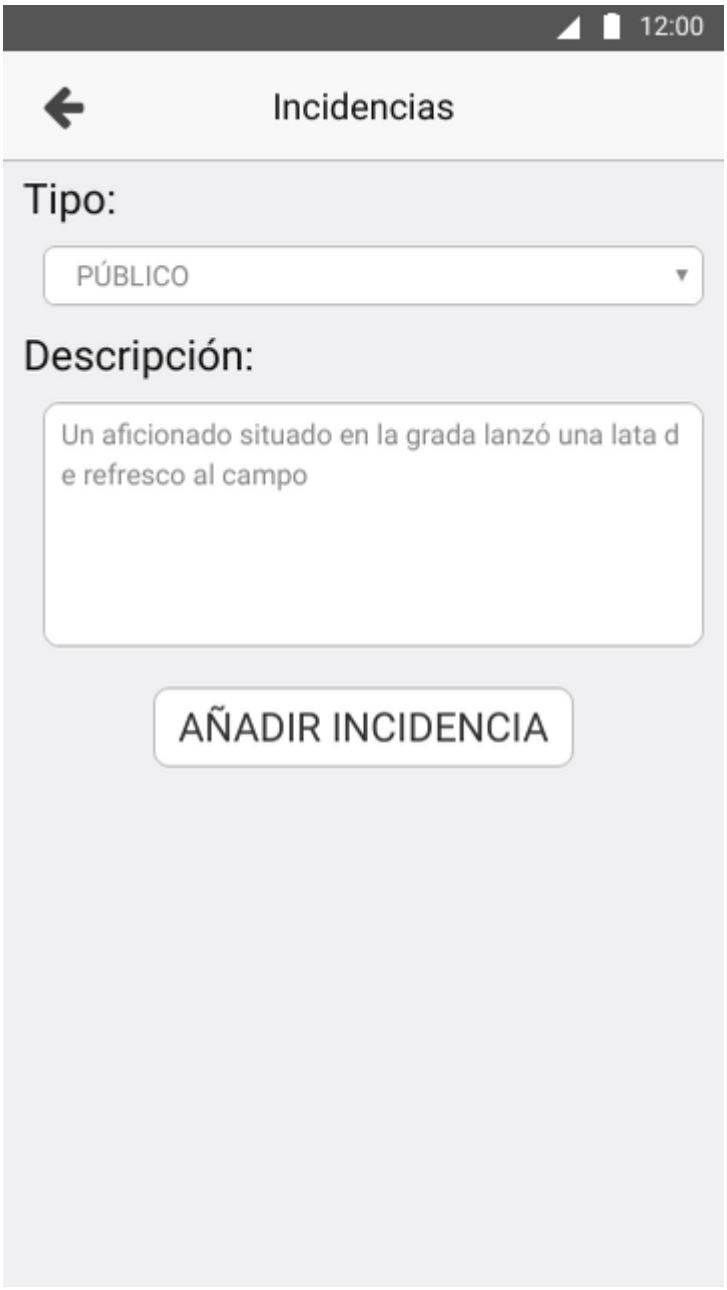

Figura C-12: maqueta correspondiente a la ventana de introducción de incidencias

D Capturas de pantalla de la aplicación

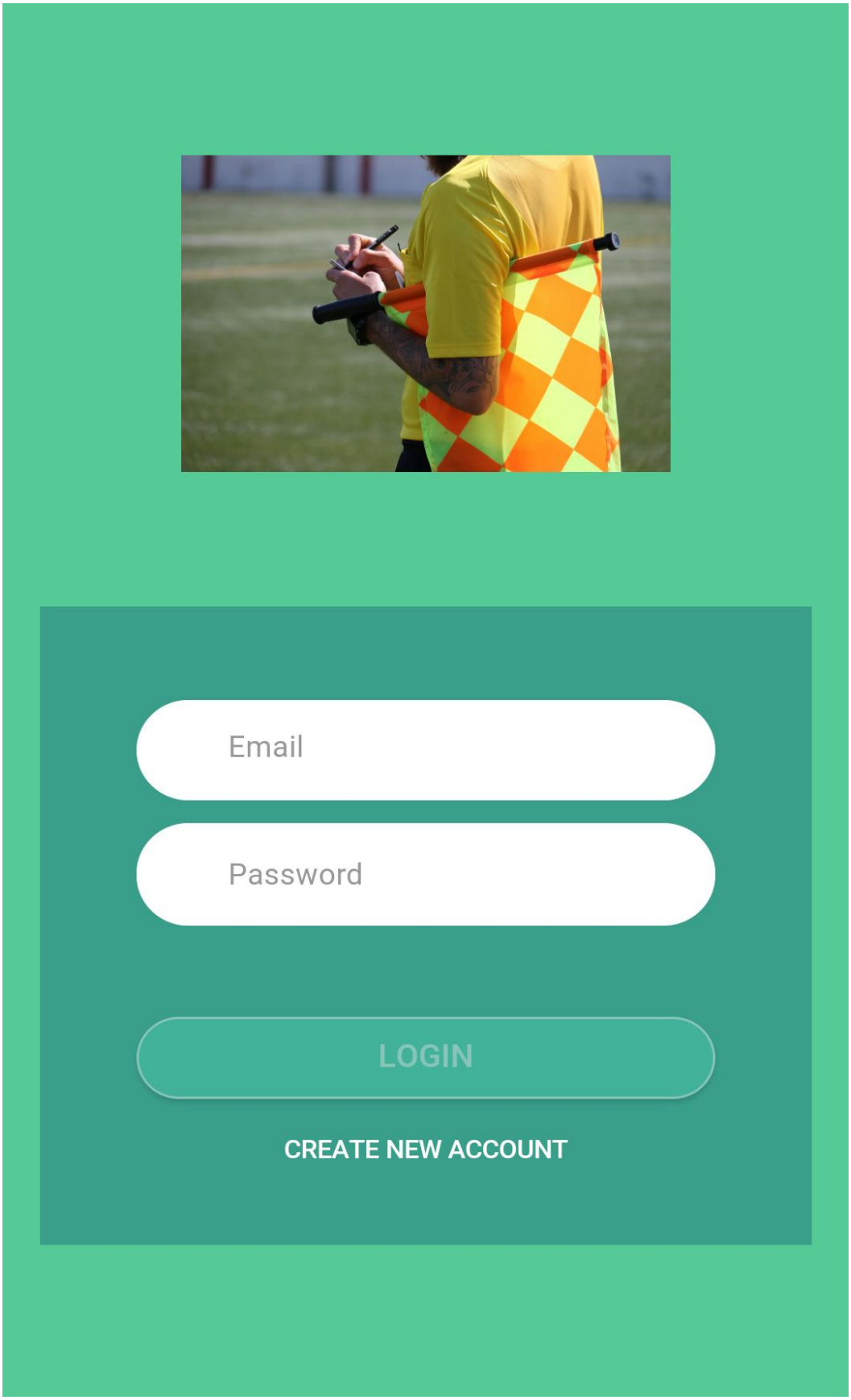

Figura D-1: pantalla de inicio de la aplicación

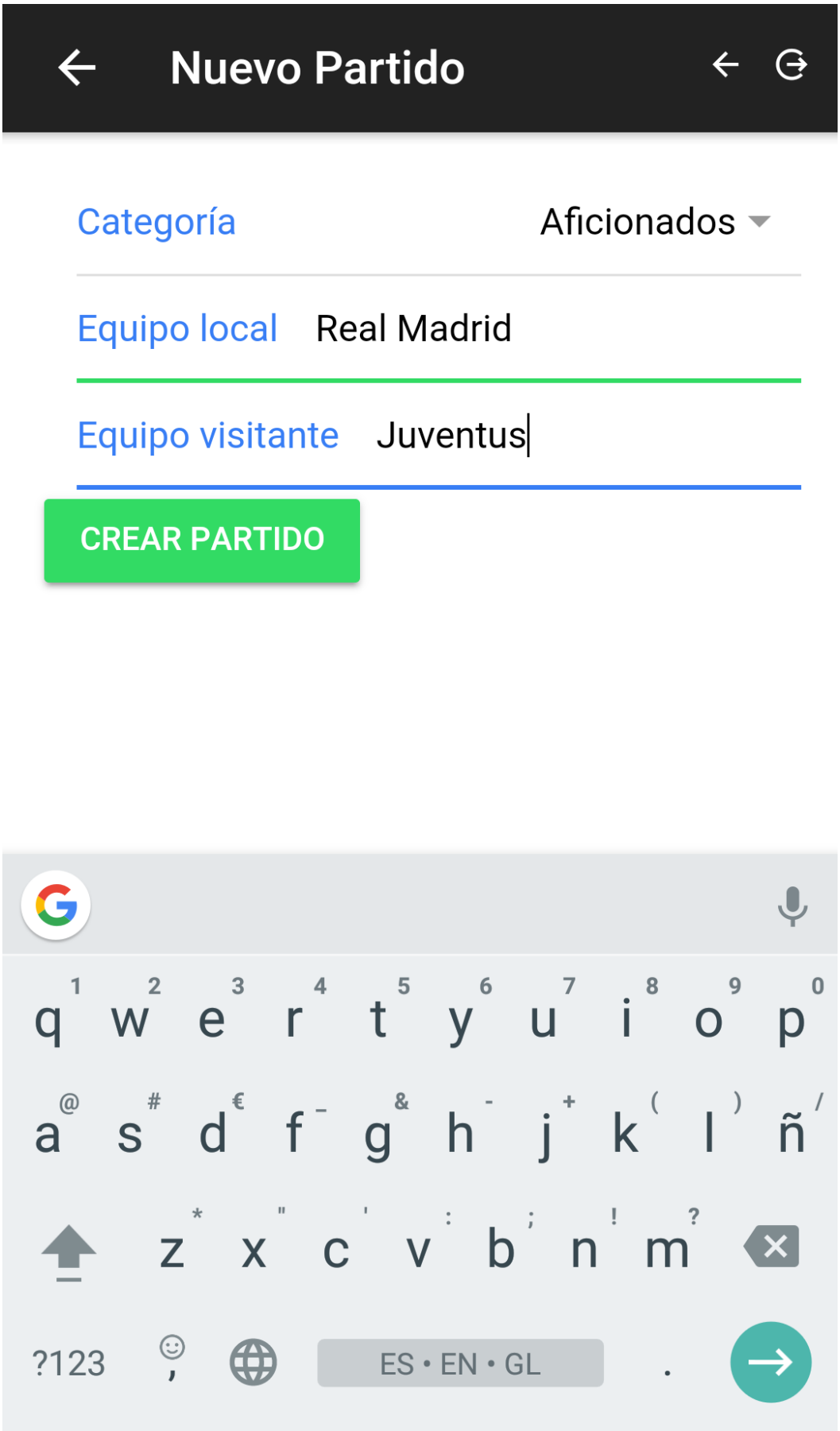

Figura D-2: pantalla de creación de un partido

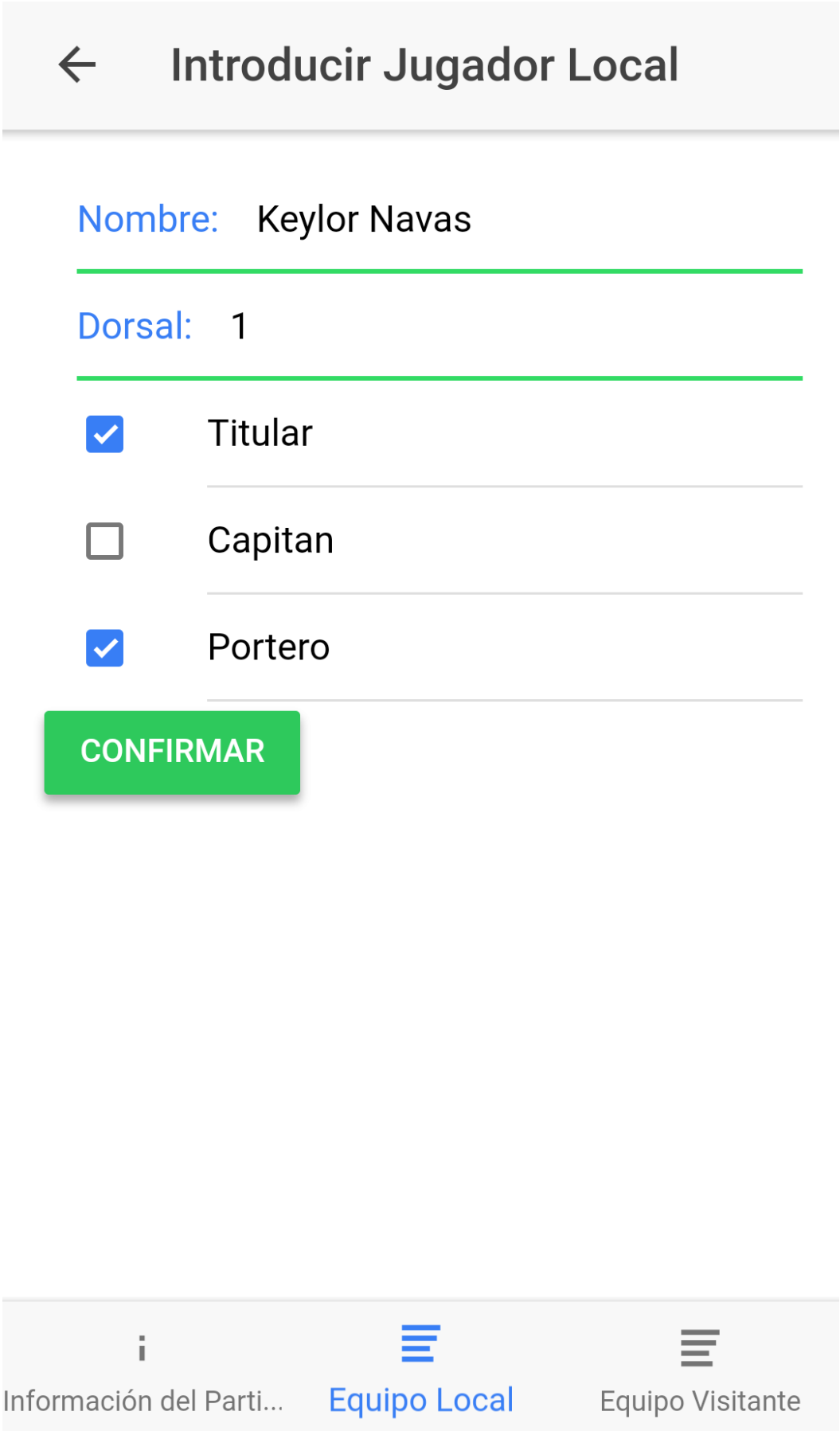

Figura D-3: pantalla de introducción de un jugador

Figura D-4: pantalla correspondiente a la alineación de un equipo antes del partido

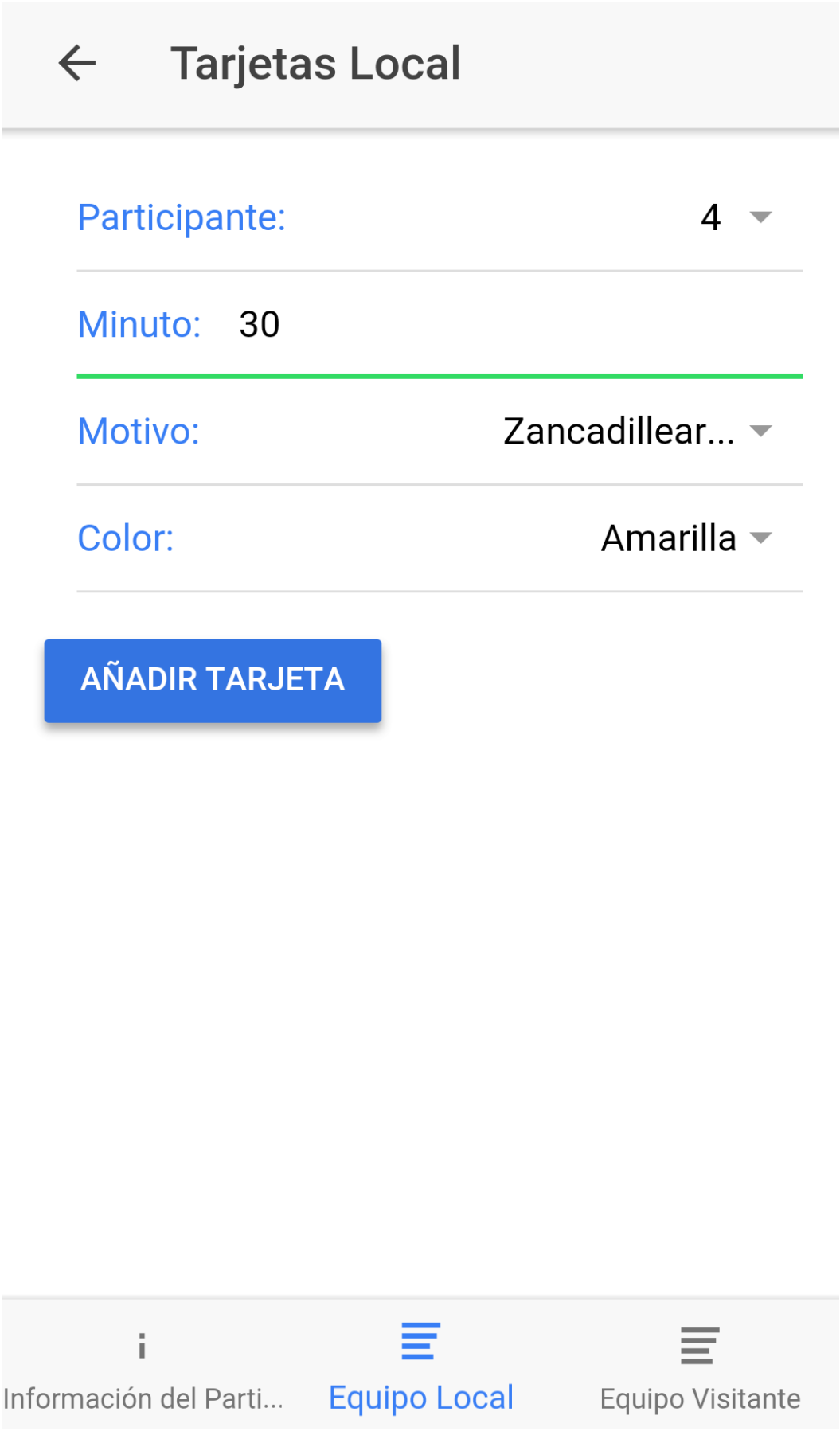

Figura D-5: pantalla correspondiente a la introducción de una tarjeta

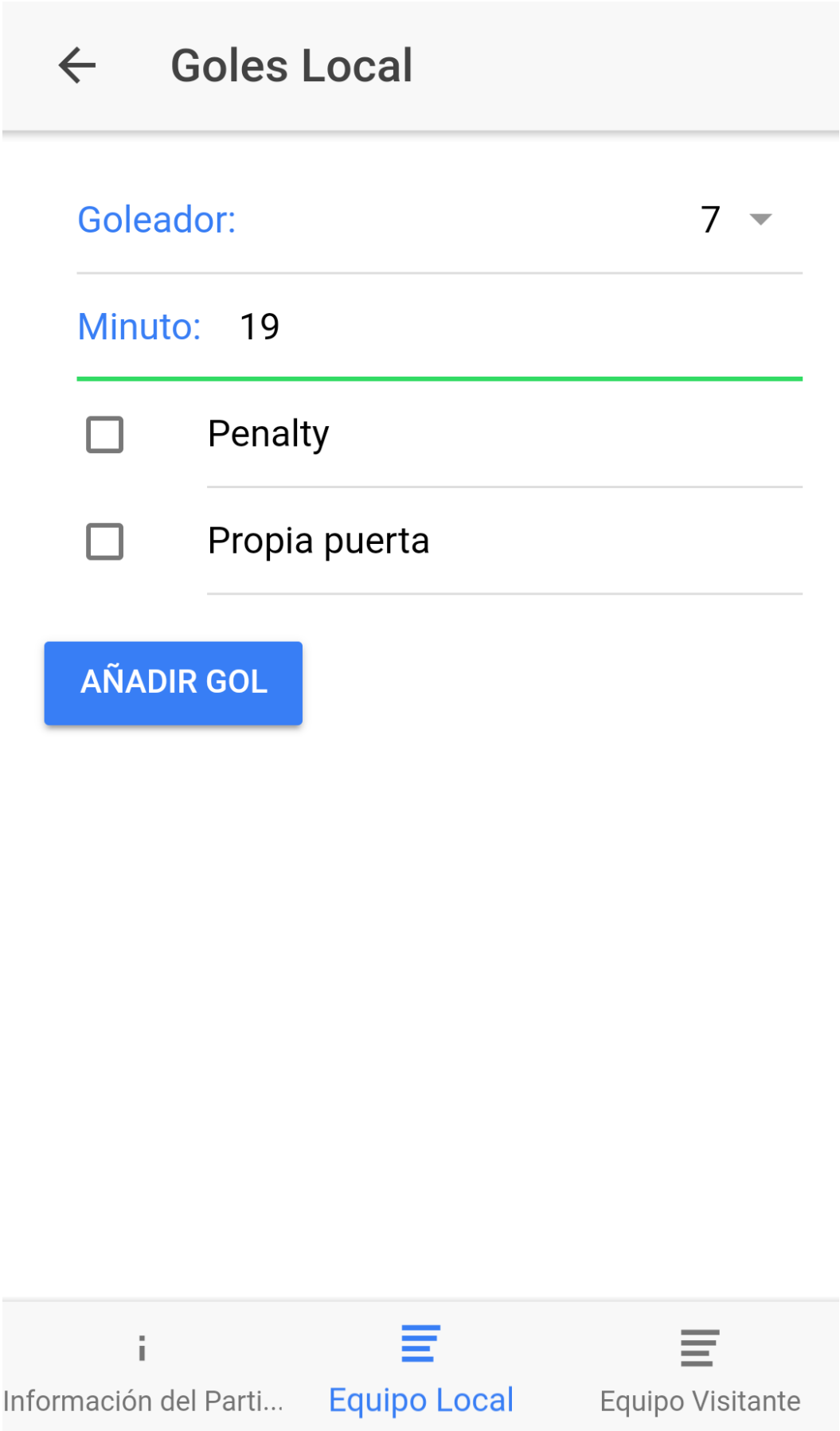

Figura D-6 pantalla correspondiente a la introducción de un gol

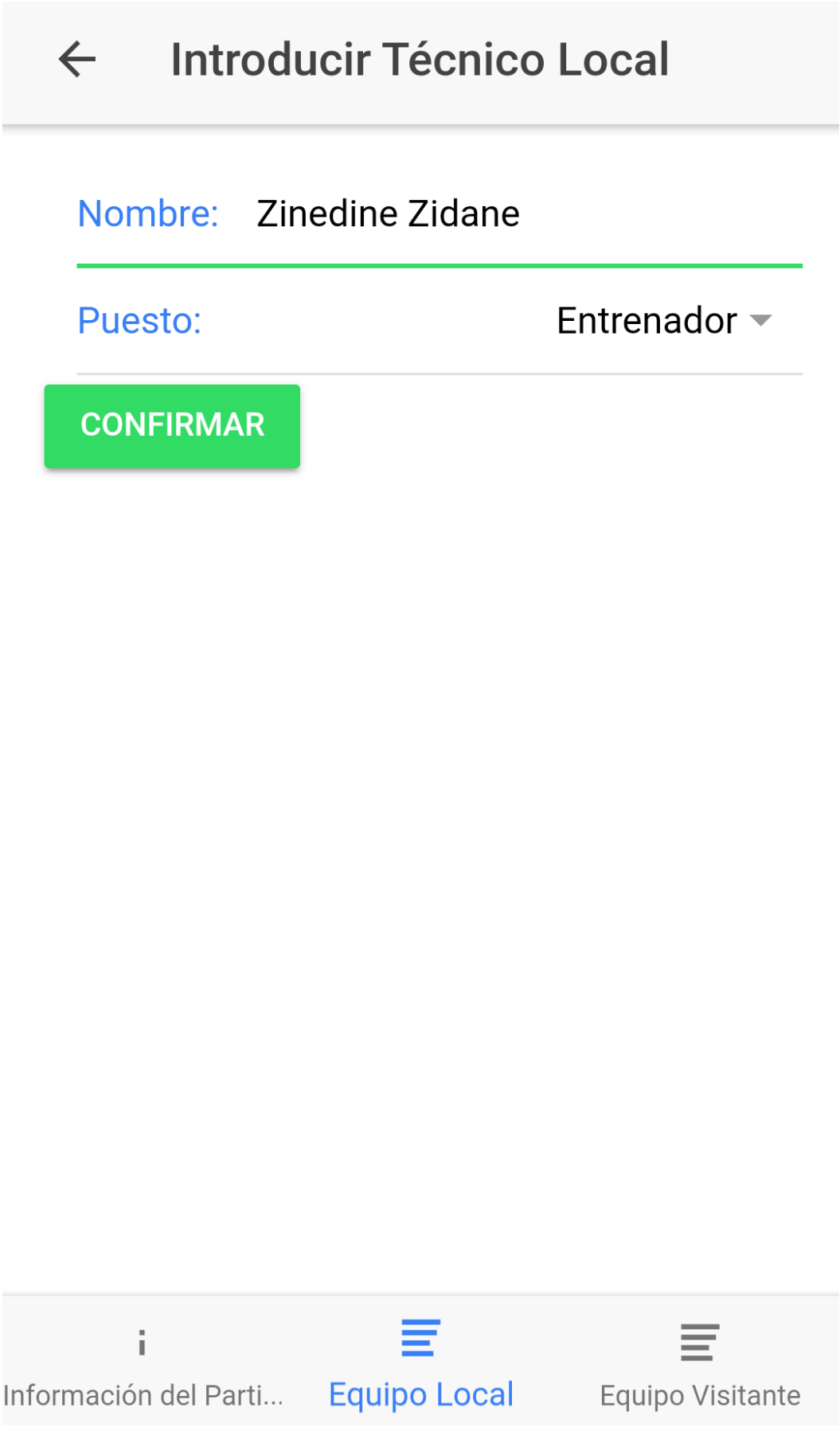

Figura D-7: pantalla correspondiente a la introducción de un técnico

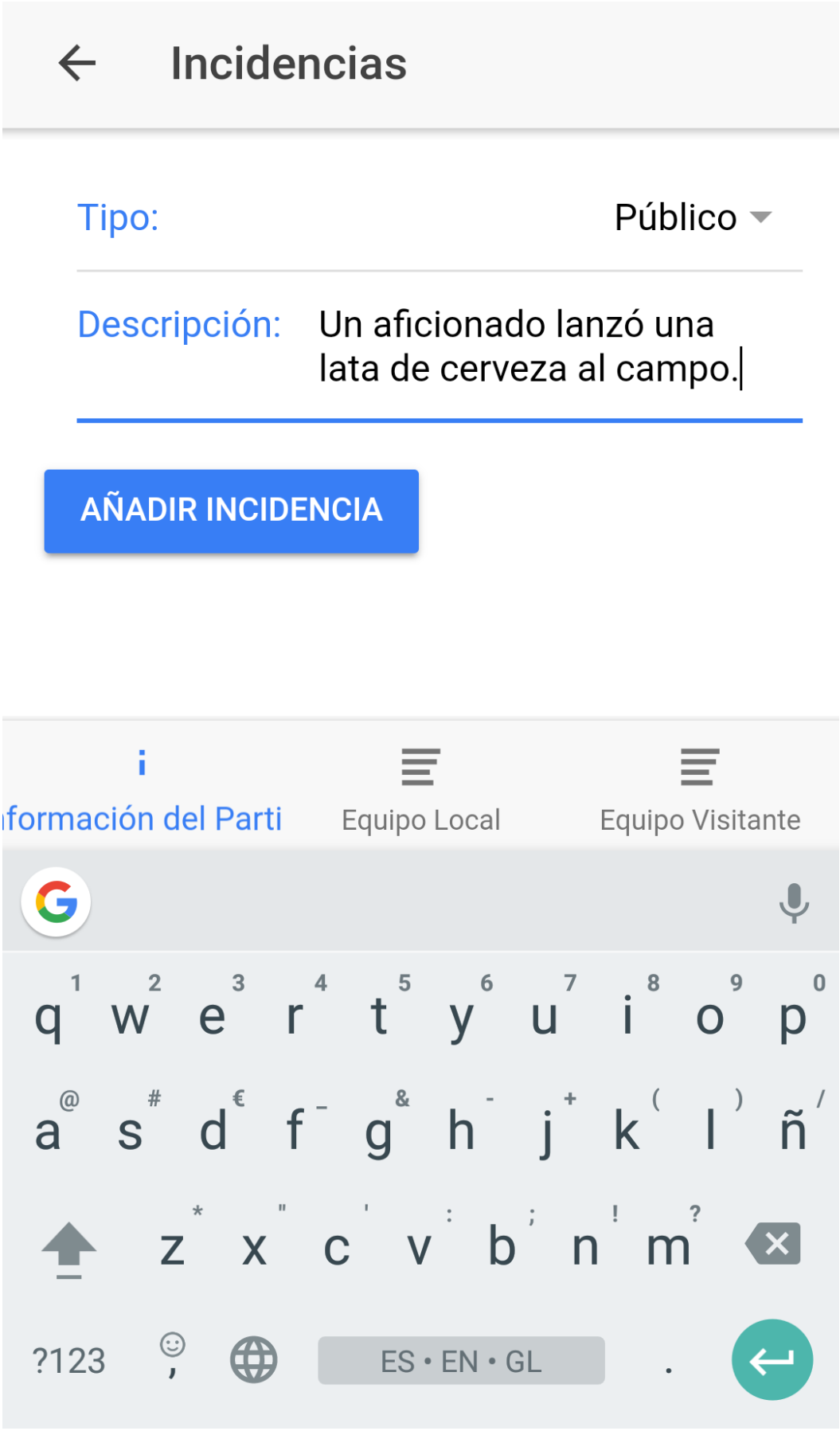

Figura D-8: pantalla correspondiente a la introducción de una incidencia